INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(54) Title: METHOD AND DEVICE FOR REDUCING SMEAR IN A ROLLED FINGERPRINT IMAGE
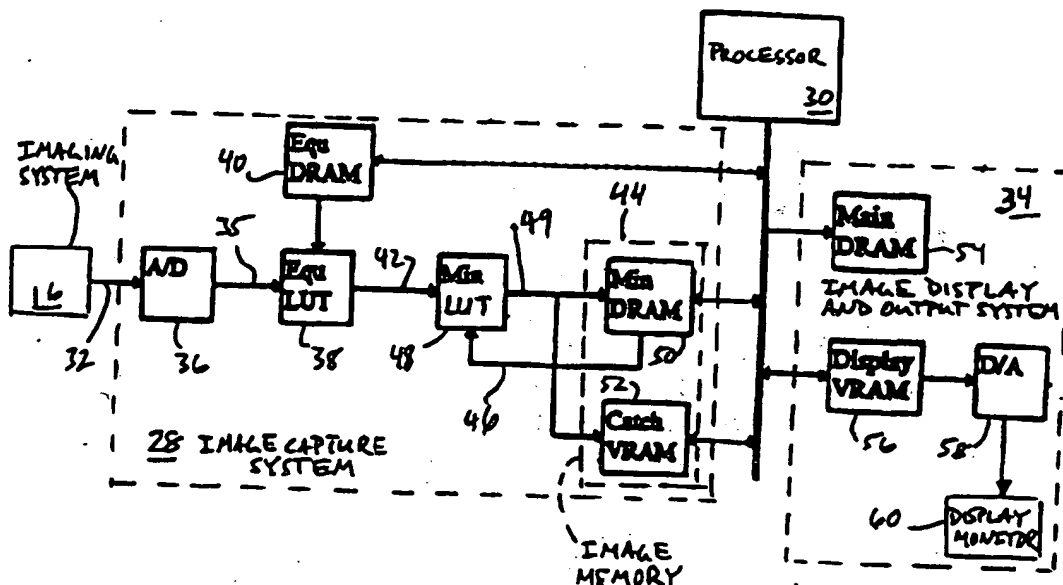
(57) Abstract

A fingerprint image capture system (28) reduces tip smear by ceasing to update a data array characteristic (44) of the rolled fingerprint image behind an advancing freeze column at least about half way from the trailing edge to the leading edge of a finger contact strip.

METHOD AND DEVICE FOR REDUCING SMEAR
IN A ROLLED FINGERPRINT IMAGE
5   Background of the Invention

The invention relates to electronic fingerprint image capture systems, and, in particular, to a method of reducing smearing in a captured rolled fingerprint image.

The traditional method of obtaining a fingerprint
10   image is to first apply ink to a subject's finger, and then to transfer the fingerprint pattern of ridges and valleys to a piece of paper by pressing the finger to the paper. The fingerprint pattern of ridges transfers to the paper, while the valleys do not. To obtain a rolled
15   fingerprint image, a side of an inked finger is placed in a designated area of the paper and then the finger is rolled to its other side on the paper.

Opto-electronic systems can capture a rolled fingerprint image without the use of ink. Typically, a
20   series of optical images of a rolling finger on an imaging surface are propagated from an imaging device and converted to digital data. A variety of methods can be used to generate a rolled fingerprint image from the digital data representative of the series of images. One
25   method is disclosed in U.S. Patent No. 4,933,976. According to this method, the propagated images are sequentially stored in the form of digital arrays of image data. Active areas of the arrays representative of fingerprint features are identified as a mathematical
30   function of the stored image data. If adjacent two-dimensional active areas have sufficient overlap, then they are merged according to a mathematical function of the data in the overlap region to form a composite array characteristic of the rolled fingerprint image. The
35   mathematical function in the composite array generating step is an average, a comparison or an average and a

- 2 -

comparison of the overlapping data in adjacent active
areas.

Another method is used in the model TP-600 system,
produced by Identix, Inc. of Sunnyvale, California. The
5   TP-600 includes an optical system having a large charge
coupled device (CCD) imager that accommodates the entire
imaging surface of an optical platen. The CCD output is
an analog signal characteristic of light and dark
patterns on the imaging surface. When a finger is placed
10 on the platen, the analog signal has lower values
(darker) for fingerprint ridge information and higher
values (lighter) for fingerprint valley information,
similar to what occurs when ink is used for
fingerprinting. The analog signal is applied to an
15 analog-to-digital (A/D) converter, the output of which is
digital image data used to update the content of an array
in image memory by means of a minimum function. Each
element in the array initially has a value that
represents the light intensity imaged at a corresponding
20 location on the platen. As the finger is rolled across
the imaging surface of the platen, the data in the image
memory is developed and updated.

The minimum function operates by preserving pixel
values in image memory that are lower than the
25 corresponding values of the incoming image data. If the
value of the current image data is lower than the
corresponding pixel value in image memory, then the lower
image data value displaces the higher value in the array.
Thus, for every location where a finger ridge contacts
30 the imaging surface a lower pixel value (darker) is
preserved in image memory. The contents of image memory
are output to peripheral devices for storing a captured
rolled fingerprint image and for real-time display of the
developing rolled fingerprint image.

- 3 -

While using the minimum function method of
acquiring a rolled print by saving the darkest intensity
value will produce a good quality print, free of
recognizable artifacts, and will be insensitive to the
5 speed with which the finger is rolled, some areas of the
print tend to have a smeared characteristic, reducing the
differentiation between ridges and valleys. This effect
occurs where the finger slides on the imaging surface
while still in contact. The smearing often occurs at the
10 tip of the finger and at the edge of the contacting area.
The use of tacky coatings on the contact surface reduces
overall slippage, but the rounded geometry of the finger
makes tip smear a continuing problem. While smearing is
found in inked prints as well as those obtained by the
15 opto-electronic system, it would be advantageous for the
opto-electronic systems to improve the clarity of the
image in the areas in which slip occurs.

Summary of the Invention

The invention provides a method of reducing smear
20 in a rolled fingerprint image represented by a rolled
image array. The method includes the step of generating
a series of frames of an optical image signal, wherein
the optical image signal includes data characteristic of
light intensities of corresponding locations of an
25 optical image, wherein the optical image includes a
fingerprint image of a finger rolling on a surface. The
method also includes determining, for each frame of the
optical image signal, a freeze column representing a line
positioned between leading and trailing edges of the
30 fingerprint image and oriented transverse to a direction
of roll of the finger. The method further includes
sequentially updating an interim array that is an
accumulation of the frames of the optical image signal
and characteristic of an interim image of a rolled

- 4 -

fingerprint. A current update of the interim array is
formed by reducing pixel values of the interim array with
a portion of the difference between the corresponding
data values from the current frame of the optical image
5   signal and the pixel values of the interim array only if
the corresponding data values of the current frame of the
optical image signal are less (characteristic of darker
features) than the corresponding pixel values of the
interim array. The rolled image array is generated by
10 transferring portions of the interim array to the rolled
image array in concert with the movement of the finger
image in the optical input signal.

      During each update cycle, a new freeze column is
determined at a position near a midpoint of a finger
15 contact area which in turn is determined from the leading
and trailing edges of the fingerprint image associated
with a current frame of the optical image signal. The
rolled image array may be initialized with a trailing
portion of a current interim array, the trailing portion
20 being interim array data behind a current freeze column
in a direction of finger roll. Each time a new freeze
column is determined by a processor in the system,
current interim array data between the current freeze
column and the previous freeze column is transferred to
25 the rolled fingerprint image array. Alternatively,
current interim array data between the previous freeze
column and data characteristic of the leading edge of the
rolled fingerprint in the interim image is transferred to
the rolled fingerprint image array each time a new freeze
30 column is determined. In both cases, the trailing
portion of the interim array behind the previous freeze
column is not used to further update the rolled image
array. Thus, the data in the rolled image array is
frozen behind a freeze column that moves in the direction
35 of finger roll and smearing in the rolled fingerprint

image due to finger movement behind that column is eliminated.

In addition to eliminating smear in the rolled fingerprint image behind the freeze column, the invention 5 preserves the benefits provided by the minimum function in merging a series of frames of the image data signal.

Brief Description of the Drawing

The accompanying drawings, which are incorporated and constitute a part of the specification, schematically 10 illustrate an embodiment of the invention and, together with the general description given above and the detailed description of the embodiments given below, serve to explain the principles of the invention.

Fig. 1 is a perspective view of an image capture 15 device.

Fig. 2 is a diagrammatic illustration of the prior art rolled fingerprint optical imaging system of the image capture device illustrated in Fig. 1.

Fig. 3 is a functional block diagram of the image 20 capture device of Fig. 1.

Fig. 4 is a functional block diagram of a portion of the image capture device of Fig. 1.

Figs. 5A-E illustrate a series of optical images of a finger rolling on a platen.

25 Figs. 6A-6E illustrate a series of images represented in the image memory shown in Fig. 3 by an interim data array. The images temporally correspond with the images illustrated in Figs. 5A-5E, respectively.

Figs. 7A-7E illustrate a series of images 30 represented in output DRAM by a rolled fingerprint image array. The images temporally correspond with the images illustrated in Figs. 6A-6E, respectively.

Figs. 8A-8E illustrate a series of images represented in display VRAM by a rolled fingerprint image

- 6 -

array. The images temporally correspond with the images
illustrated in Figs. 6A-6E, respectively.

Detailed Description of the Invention

A smear reduction method for reducing the effects
5 of smearing in rolled fingerprint images is provided.
Referring to Fig. 1, the smear reduction method may be
incorporated into the operation of a model TP-600
fingerprint capture device 10, manufactured by Identix,
Inc., the assignee of the subject matter of this
10 application.

The TP-600 includes separate imaging systems for
obtaining a rolled fingerprint image and for obtaining a
plain, or slap image. The plain fingerprint imaging
system 12 produces an analog signal representing the
15 image of one or more fingers pressed to a plain print
platen 14, and a rolled fingerprint imaging system 16
that produces an analog signal representing the image of
a finger 18 being rolled across a rolled print platen 20.
Referring now also to Fig. 2, each imaging system
20 includes an illumination source 22, optics 24, and a
large CCD imaging device 26 that accommodates the entire
image from the platen surface.   In the described
embodiment, the CCD imaging device 26 for the rolled
fingerprint image is a model TC217 CCD imaging array,
25 available from Texas Instruments, Inc. of Dallas, Texas.
Although only one mirror is shown in Fig. 2, optics 24
actually includes a combination of prisms, mirrors, and
lenses selected and arranged to bring the image from the
platen surface to the CCD imaging device 26. The plain
30 print platen 14 is wider than the rolled print platen 20
to accommodate four fingers rather than one finger on its
surface, and its optics 24 are arranged differently to
accommodate the larger imaging surface. The purpose of
each system is to present a fingerprint image at the

- 7 -

surface of a CCD imaging device when a finger is applied
to the imaging surface of the platen.

Referring now to Fig. 3, the output of the CCD
imaging device 26 is an analog signal 32 which is applied
5   to an image capture system 28. The illumination and
imaging, and the CCD output convention employed present
an image signal that has lower values (darker) for ridge
information and higher values (lighter) for valley
information.

10   A processor 30 is used to manage the transport of
data between and through each functional element of the
system and to perform other "housekeeping" functions such
as writing text to an image display monitor 60 in the
image display and output system 34, intercepting switch
15   closures and performing system start-up and shut-down
operations. As will be described in greater detail
below, the processor 30 also actively manages the
processing of image data as the finger is rolled on the
platen surface in forming a rolled fingerprint image.
20   For the described embodiment, a graphic processor
manufactured by Texas Instruments, Inc., part number
TMS34020, is used. This particular processor supports
special functions for processing two-dimensional arrays
in memory. A copy of the source code in C language for
25   operating the TP-600 is included in the microfiche
appendix.

Referring now also to Fig. 4, the analog signal 32
from the CCD imaging device 26 of the imaging system 16
is applied to an analog-to-digital (A/D) converter 36
30   that is part of image capture system 28. Because the
illumination of the fingerprint is not uniform in the
scanner, the data values of the A/D output digital data
35 are individually scaled by an equalization look-up
table (Equ LUT) 38 according to table values stored in
35   the equalization memory (Equ DRAM) 40. The stored

- 8 -

reference values correspond to an image of the surface of
a blank platen 20, smoothed to eliminate noise and
surface contamination.

The output from Equ LUT 38 is an optical image
5   signal 42 in the form of a stream of digital data that
can be grouped in frames. The data have values which are
characteristic of the light intensity of corresponding
locations of the imaging surface of the platen 20. Each
frame corresponds to an image of the platen at a
10   different time. The data values are updated about
fifteen times a second. Thus, about 25-35 frames of
optical image signal 42 are generated during the time it
takes for the finger 18 to roll across the surface of the
platen 20.

15   The optical image signal 42 is used to update the
content of an image memory 44, which holds a 968 X 968
pixel interim data array, by means of a functional
element identified as a "minimum function" look-up table
(Min LUT) 48. This size array is sufficient to produce
20   an image with a resolution of 600 dots per inch. The
inputs to Min LUT 48 are the A/D converter output 35 as
modified by Equ LUT 38, which is the current optical
image signal 42, and the corresponding old interim data
array pixel values 46 which are to be updated. The
25   "latest value" is input from the current frame of optical
image signal 42 and the "old value" is input from the
current interim data array, as most recently updated by
the previous frame of optical image signal 42.

In the simplest implementation, the Min LUT 48
30   computes $F^n_{i,j}$, the new pixel value 49 of the interim data
array at row i and column j, as a minimum, $F^n_{i,j}$ =
$min(I^n_{i,j}, F^{n-1}_{i,j})$, where I^n is the input datum value of the
nth frame from the Equ LUT 38 and $F^{n-1}_{i,j}$ is the feedback 46
from the image memory 44 from the preceding frame. The
35   output signal 49 of Min LUT 48, for each pixel of interim

- 9 -

data array, is the lower value of its two inputs, as suggested by its name. For each datum output by the A/D converter 36 (as modified by Equ LUT 38 to form optical image signal 42), the corresponding pixel of interim data array in the image memory 44 is updated.

It is not necessary to store the digital data 35 output from A/D converter 36 and the optical image signal 42 from Equ LUT 38 as arrays before being processed by Min LUT 48. The values of output data 49 from Min LUT 48 used to update the interim data array depend only on the corresponding datum values of the optical image signal 42 and on the old corresponding pixel values 46 of interim data array. For every location where a finger ridge contacts the imaging surface of the platen 20, a lower pixel value (darker) is preserved. The result of this technique is that as the finger 18 is rolled across the imaging surface of platen 20, an interim rolled fingerprint image is constructed in image memory 44. This process has been found to eliminate artifacts such as fingerprint features or discontinuities that are not part of the true fingerprint.

When performing a capture of a rolled fingerprint image, the interim data array in image memory 44 must be initialized since feedback is involved. One way to initialize image memory 44 is to set all the pixel values to a maximum value. Then the interim data array in image memory 44 will immediately reflect any data that is input in the next frame. In another embodiment, interim data array can be initialized by setting up Min LUT 48 as a straight-through function such that its output is the same as the optical image signal 42 input from the Equ LUT 38. The first frame of optical image signal 42 can then update the image memory 44 independently of what is already stored.

- 10 -

As the finger 18 is rolled, the edges of the
contact area of the finger on the platen may move fairly
rapidly with respect to the frame update rate. This may
cause some discontinuities to occur between the
5 interlaced fields of the video. Similarly, the tip of
the finger often slides as it contacts the platen,
causing discontinuities. To resolve this problem, the
function loaded into the Min LUT 48 can be modified from
a strict minimum such that when the input datum value I^n
10 is less than the previous interim array value F^{n-1}, the
interim array value is reduced by a portion of the
difference, $F^n = F^{n-1} - K*(F^{n-1} - I^n)$, where K is a factor
less than or equal to one that sets how fast the value in
a pixel may change. Noticeable improvement in the image
15 quality can be obtained with K in a range of 0.25 to 0.5.
For the described embodiment, K is set to approximately
0.33. This function causes the conditions of concern to
appear as gray smears instead of jagged discontinuities,
since the conditions are often only present for a small
20 number of frames.

The Min LUT 48 has a 64 Kb x 8 SRAM and registers
to pipeline the input and output. A 64 Kb address space
requires 16 address lines. The two 8-bit inputs to the
Min LUT 48 are tied to 8 address lines each. Thus, for
25 each set of the two input values there is one
corresponding location in the SRAM which contains the
desired value to be output. This implementation is very
unrestrictive, since any function can be implemented in a
tabular form. The different functions to be used in the
30 Min LUT 48 are typically precomputed and stored in a Main
DRAM 54 and then loaded into the SRAM when needed.

Image memory 44 includes two redundant memories,
Min DRAM 50 and Catch VRAM 52. They independently and
simultaneously hold the same interim data array for
35 transfer to image display and output system 34. Image

- 11 -

display and output system 34 includes a main output
memory 54 (located in Main DRAM) and a display memory 56
(located in Display VRAM) that receive data transferred
by processor 30 from Min DRAM 50 and Catch VRAM 52,
5 respectively. The display memory 56 is used to provide
information to the operator in real-time. The display
memory 56 receives image information along with
fingerprint placement cursors and text information
providing instructional information to the operator. The
10 display memory 56 typically contains less information
than that contained in image memory 44 or output memory
54 for reasons of data efficiency, display raster size,
and other display limitations. The output memory 54 does
not contain text information and finger placement cursor
15 information. This memory contains all the high quality
image data.

The interim data array in image memory 44
represents an interim rolled fingerprint image, and could
be transferred in its entirety with each frame to output
20 memory 54 or display memory 56 to form a rolled
fingerprint image array. This is the method of the prior
art TP-600. However, if the finger 18 slips on the
imaging surface of the platen 20 when the interim data
array is being formed, then the rolled fingerprint image
25 will appear smeared, similar to what happens with the ink
and paper method of obtaining a rolled fingerprint image.
The smear reduction method of the invention reduces
smearing in the rolled fingerprint image by transferring
to output memory 54 and display memory 56 only a selected
30 portion of the interim data array 46 with each video
frame.

Typically, an operator will preview the finger
image prior to entering a capture mode to obtain the
rolled fingerprint image. In order to place the finger
35 20 properly on the platen 18, it is helpful to be able to

- 12 -

center the finger while viewing the image of the finger
on a display monitor 60. The operator sets Min LUT 48 to
the straight-through function and rolls the finger to one
side to prepare for the capture of the rolled image. The
5 image displayed is then not a rolled image but a direct
image of the finger 18 on the platen 20. Since the
capture mode is entered after a scan button is pressed,
the data in the image memory 44 at the end of the preview
mode serves to initialize that memory for the capture.

10   Referring now to Fig. 5A, the first frame of
optical image signal 42 after capture mode is entered
represents an optical image 62a of the surface of the
platen 20, including an image of contact area 64a of the
finger 18 on platen 20. (The cross-hatching in the
15 drawing indicates fingerprint features.) In Fig. 5B,
contact area 64b is to the right of the location of
contact area 64a, indicating that the finger 18 has
rolled to the right. The contact area 64 continues to
move incrementally to the right in Figs. 5C and 5D. In
20 Fig. 5E, the contact area 64e has shrunk in size from
previous contact area 64d, as the finger 18 is lifted
from the platen 20.

        As the capture mode is entered, the Min LUT 48 is
restored to the modified minimum function, as described
25 above. Referring now also to Fig. 6A, the most recent
frame of optical image signal 42 that was passed through
Min LUT 48 becomes an initial frame of interim data
array, which is characteristic of an interim image 66a
that includes interim rolled fingerprint image 68a.
30 Interim rolled fingerprint image 68a, in this embodiment,
is the same as the corresponding contact area 64a
illustrated in Fig. 5A. Alternatively, interim data
array can be initialized with all high pixel values,
indicative of a blank, illuminated platen (not shown).
35 Min LUT 48 can then update interim data array using the

- 13 -

modified minimum function, with a first frame of optical
image signal, represented by optical image 62a, as one
input and corresponding pixel values of the "blank"
interim data array as the other input. The resulting
5 interim data array is essentially the same in either
case.

The interim data array is next updated when the
second frame of optical image signal 42, represented by
optical image 62b in Fig. 5B, is processed through Min
10 LUT 48 with corresponding pixel values 49 of the interim
data array, represented by the previous interim image
66a. The updated interim data array is now
characteristic of an interim image 66b that includes
interim rolled fingerprint image 68b, illustrated in Fig.
15 6B. Similarly, Fig. 6C illustrates interim image 66c and
interim rolled fingerprint image 68c, which are
represented by interim data array in image memory 44
after being updated with the next frame of image signal
42, which is represented by optical image 62c,
20 illustrated in Fig. 5C. Figs. 6D and 6E illustrate
respective interim images 66d, 66e and interim rolled
fingerprint images 68d, 68e represented by subsequent
updates to interim data array.

Contact area detection can be done in many ways.
25 One method is to finely segment the optical image signal
42 and then compute the variance of the data values in
each of the segments. A segment with a small variance is
considered to have no contact. Another way is to
threshold each data value and to consider it contacted
30 when the value drops below a fixed level. This is
acceptable when the image background is equalized by Equ
LUT 38 since then a fixed level corresponds to a
consistent degree of contact across the complete image.

The bottom or tag bit (bit 0) of interim data
35 array in image memory 44 is allocated to the function of

- 14 -

indicating contact.   Since Min LUT 48 is completely
flexible as to what is programmed into it, the function
for the tag bit 0 may be handled separately from bits 1-
7.   The tag bit is set to 1 if the input value to Min LUT
5   48 from optical image signal 42 is less than a threshold
T.   The information reflecting the contact area 64 is
thus available in the tag bit 0 of the Min DRAM 50 and
Catch VRAM 52 as a binary images 70a-70e, which have
outlines indicated in Figs 6A-6E, respectively, by dashed
10   lines.   The accumulated gray-scale interim images 66a-66e
are available in the upper bits 1-7 of the Min DRAM 50
and in the Catch VRAM 52 of image memory 44.   It will be
understood that the processor 30 can determine the binary
images 70a-70e even when Min LUT 48 is in preview mode in
15   which the optical image signal is passed through to image
memory 44.

        The contact area 64 of the fingerprint represented
by each frame of optical image signal 42 can be modeled
most simply by a contact strip 72, with a left edge 74
20   and a right edge 76.   The contact area 64 usually has a
convex perimeter, but we have found it acceptable to
consider contact strip 72 to be rectangular-shaped, with
the left edge 74 as the column at the left-most edge of
the contact area 64 and the right edge 76 as the column
25   at the right-most edge of the contact area 64.   The
processor 30 determines the right edge 76 and left
edge 74 of the contact strip 72 from binary contact
image, generally referred to by reference numeral 70, in
the Catch VRAM 52.   This is done in a time frame
30   comparable to the frame update rate in order to keep up
with the rolling finger.

        One way to determine the contact strip 72 is to
examine one row of tag bits across the center of the
binary contact image 70.   The left-most tagged pixel is
35   found by searching for the first tagged bit in the row

- 15 -

from the left edge of interim data array, and the right-most tagged pixel is found by searching for the first tagged bit in the row from the right edge of interim data array. The left edge 74 and right edge 76 of the contact

5 strip 72 are then identified with the left-most tagged pixel and the right-most tagged pixel.

Dirt or contaminants present on the platen 20 can cause isolated pixels out of the contact area 64 to be tagged in forming the binary contact image 70. The

10 fingerprint is composed of ridges which may align with the line being checked such that a valley will confuse the location of an edge. These problems can be reduced by examining a vertical band 78 that includes a number of horizontal lines near the center of the binary contact

15 image 70. For example, a vertical band 78 of 10 lines spaced 4 lines apart across the middle of the binary contact image 70 can be used.

Using only the left-most tagged pixel as the left edge 74 of the contact strip 72, even when using more

20 than one line near the center of the binary contact image 70, can still be too sensitive to the presence of dirt and falsely indicate contact or distort the finger image at the edges of the contact area 64. To mitigate this problem, in one embodiment, a number of tagged pixels are

25 counted from the left side of the binary contact image 70 before establishing a column as the left edge 74 of the contact strip 72. The processor 30 determines the right edge 76 of the contact strip 72 in a similar procedure. The left and right edges 74, 76 of the contact strip 72

30 are established as the 10th tagged pixel in from the outside edges.

The processor 30 supports a special mode which allows processing operations to be performed during a two-dimensional block transfer. One of the operations is

35 a logical OR. Thus a number of rows may be transferred

- 16 -

to one final row while performing a logical OR. The
destination row thus provides an indication of the
contact strip over a band 78 instead of a single line.

By identifying the contact strip 72, the processor
5 30 is able to perform several other new functions. It
keeps track of the left edge 74 and right edge 76 of the
contact strip 72, and determines when the finger 18 is
placed upon the blank platen 20, when the finger 18 is
rolled and in what direction, and when the finger 18 is
10 lifted from the platen 20.

When the finger 18 is initially placed on the
blank platen 20, the left edge of the finger contacting
the platen will be beyond the right edge in a direction
from right to left. As the finger 18 is placed down, the
15 contact strip 72 will have a positive width between the
right edge 76 and the left edge 74. As long as the left
edge 74 of the contact strip 72 keeps going left and the
right edge 76 of the contact strip 72 keeps going right,
it can be considered that the finger 18 is still in the
20 process of being placed on the platen 20, with the
contact strip 72 growing. If the finger 18 is already on
the platen 20, the contact strip 72 will start at a
positive value. This is the most common situation, when
the preview mode is used to place the finger 18 and roll
25 it back to the starting position. The finger 18 usually
is not raised again before the capture mode is started.

The processor 30 determines that the rolling of
the finger 18 has begun when one edge of the contact
strip 72 starts to go inward instead of outward. For
30 example, when the left edge 74 starts to go right, as
illustrated in Figs. 6A-6D, the processor 30 determines
that the finger 18 is being rolled right, in which case
the right edge 76 is the leading edge and the left edge
74 the trailing edge of the rolling finger. If, instead,
35 the right edge 76 begins to go left, the processor 30

- 17 -

determines that the finger 18 is being rolled left, in
which case the left edge 74 is the leading edge and the
right edge 76 is the trailing edge of the rolling finger.
A small tolerance for jitter is allowed by determining
5 that rolling is begun when the left edge 74 (or right
edge 76) of the contact strip 72 moves back from its most
extreme position by a small number of pixels, nominally
5. If the left edge 74 is moving right and the right
edge 76 moving left for a predetermined number of frames,
10 then the processor 30 determines that the finger 18 is
being lifted from the platen 20.

        The processor also determines from each frame of
image signal 42 a freeze column 80 which corresponds with
a position in the contact area 64, or binary image 70,
15 located between the left edge 74 and right edge 76 of the
contact strip 72 for each frame. In one embodiment, the
freeze column 80 corresponds to a position located
approximately half the distance from the trailing edge to
the leading edge. In another embodiment, the freeze
20 column corresponds to a position located more than half
the distance from the trailing edge to the leading edge.

        Instead of transferring the entire interim data
array to output memory 54 only after the finger 18 is
finished rolling across the image platen 20, as was done
25 with prior art embodiments of the TP-600 device, the
porocessor 30 transfers a portion of the interim data
array to output memory with each new frame as the finger
rolls. The processor 30 ceases to update a portion of a
rolled fingerprint image array in output memory 54 behind
30 the freeze column determined from the preceding frame of
optical image signal 42. The freeze column 80 moves in
increments from frame to frame with the right and left
edges 76, 74 of the contact strip 72 in the direction of
roll. Since the data in the developing rolled
35 fingerprint image array behind the freeze column 80 is

- 18 -

not updated, any image smearing that develops in a
trailing portion of the interim image 62 represented by
the interim data array does not show up in the rolled
fingerprint image array.

5   Referring now to Fig. 7A, an output rolled
fingerprint image array in output memory 52 is
initialized with high pixel values indicative of a blank
background 82a. Referring now also to Fig. 7B, when the
finger 18 starts to roll, the processor 30 updates the
10 output rolled fingerprint image array in output memory 54
by transferring a trailing portion of interim data array
from image memory 44. The trailing portion can be, e.g.,
the trailing portion of interim data array characteristic
of interim image 66b, starting with a column
15 corresponding to the trailing edge 74 of the contact
strip 72, up to and including the freeze column 80
determined from the current optical image signal. Output
rolled fingerprint image array at this point is
indicative of rolled fingerprint image 82b, which
20 includes transferred portion image 84b. As the finger 18
rolls, the processor 30 updates the output memory 54 to
keep up with the position of the approximate center of
the moving contact strip 72.

          Each subsequent update to the rolled fingerprint
25 image array in output memory is a portion of interim data
array block-transferred from image memory 44. In one
embodiment (see Figs. 8A-8E and related discussion
infra), the transferred portion of a current interim data
array is characteristic of the interim rolled fingerprint
30 image 68 up to approximately the leading edge, i.e., up
to a column of the interim data array corresponding to
the leading edge of the current contact strip 72. In
another embodiment illustrated in Fig. 7C-7D, the
transferred portion of the current interim data
35 array from image memory 44 is narrower, and extends only

- 19 -

up to approximately the freeze column 80 of the current
optical image signal. Fig. 7C illustrates output rolled
fingerprint image 82c, with transferred portion image
84c, after a corresponding portion of the current update
5 of interim data array (see Fig. 6C) is transferred to the
output rolled fingerprint image array. The transferred
portion of interim data array in this instance includes
all data to the right of freeze column 80b, i.e. in the
direction of finger roll, up to and including freeze
10 column 80c. Similarly, output rolled fingerprint image
82d, illustrated in Fig. 7D, is represented by the output
rolled fingerprint image array subsequent to a portion of
a subsequent update and interim data array (see Fig. 6D)
being transferred. The transferred portion includes all
15 data to the right of freeze column 80c up to and
including freeze column 80d.

        For the final update to the rolled fingerprint
image array in output memory 54, the transferred portion
extends from a column of interim data array corresponding
20 to the freeze column determined from the preceding
optical image signal 42 to at least the column of interim
data array corresponding to the most extreme position of
the leading edge of the contact strip 72. In all cases,
the transferred portion of the current interim data
25 array is characteristic of the interim rolled fingerprint
image 68 forward from the line represented by the freeze
column 80 derived from the preceding optical image signal
42. For example, the final transferred portion,
characteristic of transferred portion image 84e
30 illustrated in Fig. 7E, is data interim data array which
is characteristic of a portion of interim image 66e
forward in the direction of finger roll from the previous
freeze column 84d.

        Thus, the processor 30 freezes the output memory
35 54 behind the moving freeze column 80, which is

characteristic of a vertical line corresponding to the
approximate center of the moving contact strip 72. No
updating of the output memory 54 occurs behind that line.
While this does not eliminate tip smear, it reduces it by
5 about 50-60%. The smearing in the main part of the
fingerprint due to the movement of the back edge of the
finger is eliminated. Since smear can still occur
between the leading edge and the freeze column 80, the
method can be improved by setting the freeze column 80 at
10 a position closer to the leading edge, nominally five-
eighths (5/8) of the distance between the trailing and
leading edges.

     The processor 30 tracks the progress of the
leading edge of the contact strip 72, which is the right
15 edge 76 in the embodiment illustrated in Figs. 5-7. When
the leading edge retreats from its farthest position by a
selected number of columns as the finger 18 is lifted
from the platen 20, the processor 30 determines that the
capture is completed, performs the final update to the
20 output memory 54, and ceases to update the rolled
fingerprint image array in output memory 54 any further.
This prevents smear as the finger 18 is lifted from the
platen 20. Thus, when the finger 18 lifts from the
platen 20 or rolls backwards, the processor updates the
25 output memory 54 with the forward portion of the interim
data array and ceases any further updates.

     It is important to note that the processor 30 only
passes though a portion of the contents of the image
memory 44 to the output memory 54 at any time. This
30 portion corresponds to a narrow strip of the interim data
array located adjacent to, but not overlapping with, the
freeze column defined by the previously passed through
data. It is also worth noting that the data used to
update the output memory 54 is not representative of a
35 raw fingerprint image. Rather, the transferred data is

representative of the interim rolled fingerprint image 68
produced by the Min LUT 48 in that narrow strip 84 since
the finger 18 began to roll.

5   A rolled fingerprint image is displayed as it is
captured. The processor moves data to the display memory
(VRAM) 56 from the Catch VRAM 52 in image memory 44. The
data is then converted to video format through a digital-
to-analog converter (D/A) 58 and output to display
monitor 60. In the embodiment depicted in Fig. 4, the
10 display monitor has a display area formed by a 720 X 720
pixel array. The processor 30 decimates the image by one
pixel column out of four and one row out of four during
the transfer from the Catch VRAM 52 to the Display VRAM
56 to fit the image into the display format.

15   Referring now to Figs. 8A-8E, the processor 30
updates the display memory 56 (in the reduced format
described above) from the image memory 44 while the
finger 18 rolls on the platen 20 to generate a display
rolled fingerprint image array characteristic of a
20 display rolled fingerprint image 86. As the finger 18 is
being placed down, i.e. in preview mode, the entire
interim data array, characteristic of interim image 66a
(which is the same as optical image 62a), is transferred
to display VRAM 56 and displayed "live." In Fig. 8A, the
25 live image is image 86a. As discussed above, when the
capture mode is entered the data in interim data
array representative of the contact strip 72 between the
left and right edges 74, 76 of the binary contact image
70 is updated. When the finger begins to roll, a portion
30 of the data in interim data array from Catch VRAM 52,
which is representative of the strip behind the current
freeze column 80b relative to the direction of finger
roll, is transferred to display rolled fingerprint image
array. This strip, shown in Fig. 8B as strip image 88b,
35 extends to the far left edge of interim image 66b and is

- 22 -

therefore representative of areas of the platen surface
that are not contacted by the finger 20. Thereafter, the
portions of data from the interim data
array representative of a strip of interim image 66 just
5 adjacent to, but not overlapping with the previous freeze
column 80b, 80c, respectively, and up to the leading edge
74 of the contact strip 72 are transferred from the Catch
VRAM 52 to Display VRAM 56. These strips are illustrated
in Figs. 7C and 7D as strip images 88c and 88d,
10 respectively. This maintains a complete image 86c, 86d,
respectively, of the developing rolling fingerprint image
in the display rolled fingerprint image array in Display
VRAM 56. Referring now also to Fig. 8E, when the capture
is deemed complete, the portion of the interim data
15 array representative of the forward strip 88e of the
interim image 66e, from the previous freeze column 80d to
the far forward edge of the interim data array is
transferred to the display VRAM 56. This last update may
also include data representative of areas to the far
20 right edge of the platen 20 not contacted by the finger
18. Each update is a portion of interim data array that
has been processed by the minimum function 48.

Many variations of the display method can be
implemented with corresponding differences in the display
25 quality. For example, whether to update blank areas to
the right and left of the contact area, or whether to
update from the output image memory or the catch memory,
are options that can be traded-off for processing
efficiency. Another alternative is not to display
30 anything forward of the current freeze column 80, for
example, if the processing time is needed.

After the capture is complete, the operator
presses a button to either reject the print or save the
print. If the print is saved, the background can be
35 whitened out to present a cleaner image. This is

accomplished by comparing the output image with the image
remaining on the platen. It is assumed that the operator
has lifted the finger before pressing the save button.
If a pixel value in the output image is below a

5 corresponding value of a remanent image (i.e. the image
of the platen without the finger) by a selected fraction
(e.g. approximately 5%), then the pixel is considered
contacted and is tagged accordingly. All pixels which
are not tagged are whitened to a consistent background

10 level. This eliminates any latent images that might be
present in the background or in the voids within the
print.

Note that, after each of the image memory 44,
output memory 54 and display memory 56 are initialized

15 with data representative of an initial image, there is no
place in the system where the optical image signal 42 or
portion of the optical image signal is actually stored.
All subsequent frames of optical image signal 42
representative of optical fingerprint images from the

20 image system 16 are processed through minimum function
48. It is only a portion of the data in each updated
interim data array in image memory 44 that is used to
update the output and display memories 54, 56.

In the embodiments described above, Min LUT 48

25 updates an interim data array in image memory 44 from the
existing pixel values of interim data array and
corresponding data from a new frame of optical image
signal 42 input through the A/D converter 18 and Equ LUT
38. The method only uses the bottom bit of each pixel in

30 image memory 44 to indicate contact. As the finger 20
rolls, the processor updates the output and display
memories 54, 56, respectively, with a portion of the
interim data array representing a forward portion of the
contact strip 72 in the image memory 44. Thus, output

35 and display rolled fingerprint image arrays are

- 24 -

respectively formed in output and display memories 54, 56. The rolled fingerprint image arrays are characteristic of a rolled fingerprint image.

In another embodiment, the interim data array may
5 be developed in image memory 44 such that the accumulation of the optical image signal 42 and therefore also the interim data array is stopped behind the freeze column 80. This could be implemented with hardware (not shown), for example with a hardware register (not shown)
10 that stores the information identifying the freeze column 80 and with controls to inhibit storage either right or left of the line, depending on the direction of rolling the finger 18. The processor 30 need only update that freeze column 80 and transfer the interim data array to
15 the output memory 54 when capture is complete.

The function of freezing the updating of the interim image array in image memory 44 can also be implemented as part of the operation of Min LUT 48. For example, one of the address inputs to the Min LUT 48 can
20 be allocated to selecting the data to be frozen in interim data array or to be updated by Min LUT 48. This bit can be controlled by a comparison of the image column with the freeze column 80, which is stored in a register updated by the processor 30.

25 The smear reduction can be improved by using a more general approach to defining the contact strip, for example determining the contact strip on a row by row basis, but this would take much more processing. The Min LUT 48 could still be used to update the interim data
30 array in image memory 44. The freeze position could be controlled on a line by line basis by storing the freeze column 80 for each line in a memory (not shown) addressed by line number. This memory could be updated by the processor 30 for each field, during the blanking periods
35 of the video or by using dual port techniques. The

- 25 -

freeze position for each line is developed from sensing
the active range for several lines through the image and
then providing a smoothed or interpolated position for
the freeze position of the intervening lines.

5   While the current implementation utilizes an
interlaced video input, the invention can also be
implemented with a camera that provides a progressive
scan, i.e., a scan which outputs only one frame, without
any interlacing of lines. This would obviate the need
10 for as much modification of the minimum function.

        Since the progress of the roll is being tracked by
the processor 30, it is possible to eliminate some of the
button pressing by the operator. This is principally
achieved by clearing the image memory 44 and restarting
15 capture automatically under certain conditions, depending
on the preferred mode of operating the system.

        As an example of this method, the operator
indicates he wants to save an image by pressing a save
button (not shown) or a save foot switch (not shown)
20 after the image capture is deemed complete as in the
method described above. To reject the print, the finger
is placed back down and rolled again. When the processor
detects that the finger is in contact again, the image
memory is cleared and the capture restarted. Note that
25 this allows the preview mode to be integrated with the
capture mode, provided that the operator lifts and
replaces the finger prior to the capture.

        The processor 30 can be configured to restart the
capture when it determines that the finger 18 changes
30 direction the first time. This corresponds well to the
normal operation of placing the finger down to center it,
rolling the finger back to one side, then rolling the
fingerprint.

        The operator assumes that he will roll all the
35 fingers in order. If he wishes to reject a fingerprint,

- 26 -

he presses a button. The operation is to keep placing a
finger down without rolling until it is centered, then to
roll the finger to one side, and then to roll the finger
for the capture. This operation can be determined from
5 the states identified for freezing the image - placing
the finger down, rolling left or rolling right, lifting
the finger. An additional criterion may be placed upon
the amount of roll to differentiate between placing the
finger and performing a complete capture.

10   It will be understood by those skilled in the art
of electronic fingerprint image capture that the imaging
system 16 can be designed in an equivalent embodiment to
provide a signal to the A/D converter 36 that indicates
fingerprint ridge features by high values and fingerprint
15 valley features by low values. The methods and devices
described above would then require only small
modifications to accommodate this change.

      It will also be understood that although the
optical image signal is described above as a data stream,
20 the optical image signal can also be formatted as an
array of pixels.

      The following appendix contains C language source
code for software for operating an Identix TP-600
fingerprint capture device. A portion of the disclosure
25 of the patent document contains material subject to
copyright protection. The copyright owner has no
objection to the facsimile reproduction by anyone of the
patent document or the patent disclosure, but otherwise
reserves all copyright rights whatsoever.

```
C-DOC                                04-18-1996 13:57 Page 2

Defined functions, SUMMARY Graphic TREES (of CALLER/CALLED files Structure)
=============================================================================

 812 FNGRCAPT.C     1 CaptionAct
 181 GSP_INIT.C     2  --load_LUTTBL_to_Mindraw
  79 GRABBER.C      3  --turn_grab_on
  46 GRABBER.C      4  --turn_grab_off
 628 FNGRCAPT.C     5  --display_fingerprint
1016 FNGRCAPT.C     6    |--display_4_3v
  61 MINIO.C        7    |--mio_size
  53 MINIO.C        8    |--mio_adr
  70 MINIO.C        9    |--mio_inc
  15 MINIO.C       10    |--mio_new
  38 MINIO.C       11    |--mio_move
                   12    | |--5vrbit
                   13    |..pnts_bvrq_aaa
1110 FNGRCAPT.C    14  --fill_even_pixels
  61 MINIO.C       15    |--mio_size ( 7 )
  53 MINIO.C       16    |--mio_adr ( 8 )
 282 GSP_INIT.C    17  --copy_image_to_Draw
 111 GRABBER.C     18  --turn_grab_on normal
  38 MINIO.C       19    |--mio_move ( 11 )
  61 MINIO.C       20    |--mio_size ( 7 )
                   21    |..display_column_4_3
 798 FNGRCAPT.C    22 CaptionAct
  44 GRABBER.C     23  |--turn_grab_off ( 4 )
 758 FNGRCAPT.C    24 PrevScanAct
  44 GRABBER.C     25  |--turn_grab_off ( 4 )
  79 GRABBER.C     26  |--turn_grab_on ( 3 )
                   27  |..command_key
 732 FNGRCAPT.C    28 PrevScanAct
  44 GRABBER.C     29  |--turn_grab_off ( 4 )
 181 GSP_INIT.C    30  |--load_LUTTBL_to_Mindraw ( 2 )
 111 USERINTF.C    31 PrintTitle
                   32  |..set_colors select_font text_width text_out
  88 FNGRCAPT.C    33 VOID
                   34  |..prescr_key_set
 135 FNGRCAPT.C    35 captscr_key_act
 108 FNGRCAPT.C    36 captscr_key_scr
 118 FNGRCAPT.C    37 capture_t
                   38  |..prescr_key_set
```

```
/*
** Filename: fngrcapt.c
** Purpose: Routines to handle the finger image capture and user interface.
** Date:    08/03/93
** Author:  Joyce Young
** Revised:
**          12/10/93 - Ellen Yu, rewrote this routine according new spec.
** History:
** 02/23/94 Joyce Young, Added the fng_num in IMAGE_ATTR_T
** 05/10/94 Joyce Young, Moved display_camera() & display_fingerprint() here
** 06/03/94 Joyce Young, adjust the offset of ROLL/PLAIN active image in
**             GREVRAM for new board
** 06/09/94 Ellen Yu, add #define PROTOTYPE difference the hardware
**             prototype board and other new revisions.
** 11/18/94 JY, delete CAPT_ABORT case
** 4/13/93 TS, fixed up for tip desmearing
*/


/* ............................ includes ............................ */
#include <gsprcg.h>
#include <gsptypes.h>
#include <gspglobs.h>
#include <stdlib.h>

#include "stdtypes.h"
#include "coorddef.h"
#include "gsp_defs.h"
#include "gsp_imgs.h"
#include "imgglobs.h"
#include "mem_addr.h"
#include "mem_alic.h"
#include "mimio.h"
#include "gsp_func.h"



/* ........................ External Functions ........................*/


extern VOID load_squref_to_EqDram (image_type_t camera);
extern VOID load_LUTTBL_to_MinBram (UBYTE *src_ptr);
extern VOID load_straight_thruLUT_to_EqDram (VOID);
extern keytype_t OSOPend_key(VOID);
extern VOID setup_display_screen (char *namep);

extern VOID copy_image_to_Dram (image_type_t camera, ULONG img_store_addr,
        int ht_bytes, int wd_words);

extern VOID CaptureScreen(IMAGE_ATTR_T, scan_type_t, char *);
extern VOID Capt_CaptScr(IMAGE_ATTR_T, short);
extern VOID Capt_PrevScr(IMAGE_ATTR_T, short);
extern VOID Misfng_MisfngScr(IMAGE_ATTR_T, short);
extern VOID Misfng_PrevScr(IMAGE_ATTR_T, short);
extern VOID Prev_CaptScr(IMAGE_ATTR_T, short);
extern VOID Prev_MisfngScr(IMAGE_ATTR_T, short);
extern VOID PrevInitScr(IMAGE_ATTR_T, short);


/* ........................ External Variables ........................ */
extern image_type_t squref_camera;



/* ........................ Function Prototype ........................*/
```

```
VOID my_mim_init(UBYTE *store_addr);
capture_t image_capture (image_type_t camera, scan_type_t fng_num,
        char *namep, ULONG img_store_addr);
VOID display_camera (image_type_t camera);
VOID display_column_4_3(MIN *source,MIN *disp,int x,int w);
VOID display_4_3v(MIN *source,MIN *dest);
VOID fill_col_4_3(MIN *dest, int col, int value);
VOID save_column(MIN *source,MIN *dest,int start, int stop);
VOID display_fingerprint (image_type_t camera, UBYTE *src_start_ptr,
        UBYTE *dst_start_ptr);
VOID fill_even_pixels(MIN *dest,int value);
capture_t CaptScanAct( IMAGE_ATTR_T image );
capture_t CaptSaveAct( IMAGE_ATTR_T image );
capture_t PrevSaveAct( IMAGE_ATTR_T image );
capture_t PrevScanAct( IMAGE_ATTR_T image );
capture_t PrevInitAct( IMAGE_ATTR_T image );


typedef enum
{
        NO_DIR,
        RIGHT_DIR,
        LEFT_DIR
} ROLL_DIRECTION;

int finger_type_check(scan_type_t fngnum,image_type_t *camera,
        HAND_TYPE *hand, ROLL_DIRECTION *deswer);


VOID (*prevscr_key_scr[])(IMAGE_ATTR_T, short) =
{
        NULL,
        Prev_CaptScr,
        NULL,
        Prev_MisfngScr,
        NULL,
        NULL
};

VOID (*misfngscr_key_scr[])(IMAGE_ATTR_T, short) =
{
        NULL,
        Misfng_PrevScr,
        NULL,
        Misfng_MisfngScr,
        NULL,
        NULL
};

VOID (*captscr_key_scr[])(IMAGE_ATTR_T, short) =
{
        NULL,
        Capt_PrevScr,
        NULL,
        Capt_CaptScr,
        NULL,
        NULL
};

capture_t (*prevscr_key_act[])(IMAGE_ATTR_T) =
{
        NULL,
```

```
        PrevScanAct,
        NULL,
        PrevSaveAct,
        NULL,
        NULL
);


capture_t (*captscr_key_act[])(IMAGE_ATTR_T) =
{
        NULL,
        CaptScanAct,
        NULL,
        CaptSaveAct,
        NULL,
        NULL
};


/* structures for defined arrays in memory */

MIM gvram;
MIM gvram_reduced;
MIM gminram;
MIM disp_window;
MIM plain_disp_window;
MIM data_store;
MIM central_rows;
MIM detect_row;
MIM check_row;


UBYTE detect_buf[1024];


/* ................................................................*/
/*
** func name: VOID my_mim_init()
** Purpose:   Initialize the memory areas for the grabber vram, the display
**            and the roll storage area.
** Returns:   None
*/
VOID my_mim_init(UBYTE *store_addr)
{
        mim_new(&gvram,((UBYTE *)GRB_VRAM_END)-GRBVRAM_WD_BYTES*26+100,
                -GRBVRAM_WD_BYTES,ROLL_WD_PIXELS,ROLL_HT_PIXELS);

        mim_new(&gvram_reduced,((UBYTE *)GRB_VRAM_END)-GRBVRAM_WD_BYTES*27+76,
                -GRBVRAM_WD_BYTES,ROLL_WD_PIXELS*3/4,ROLL_HT_PIXELS);

        mim_new(&gminram,((UBYTE *)MIN_DRAM_END)-MINDRAM_WD_BYTES*26+100,
                -MINDRAM_WD_BYTES,ROLL_WD_PIXELS,ROLL_HT_PIXELS);

        mim_new(&disp_window,((UBYTE *)OPY_VRAM_BASE)+OPYVRAM_WD_BYTES*6+12,
                OPYVRAM_WD_BYTES,720,720);

        mim_subset(&disp_window,&plain_disp_window,0,732-IMG_PLAIN_H,
                IMG_PLAIN_W-12,IMG_PLAIN_H);

        mim_new(&data_store,store_addr,ROLL_WD_PIXELS,
                ROLL_WD_PIXELS,ROLL_HT_PIXELS);

        mim_new(&central_rows,mim_adr(&gvram,0,400),5*mim_inc(&gvram),
                ROLL_WD_PIXELS,25);
```

```
        aia_new(&detect_row,detect_buf,0,ROLL_NO_PIXELS,1);

)


/* ...........................................................*/
/*
** Func name: VOID my_aia_init()
** Purpose:   Image capture process from the given camera
**            (Roll = 0, Plain = 1).
** Returns:   type in the typedef capture_t
*/
capture_t image_capture (image_type_t camera, scan_type_t fng_num,
        char *namep, ULONG img_store_addr)

    {

        IMAGE_ATTR_T  image;
        keytype_t key_pressed;
        capture_t ret;
        UBYTE *src_start_ptr, *dst_start_ptr;

        int far_r_edge,far_l_edge;
        int r_edge,l_edge;
        int s_w,s_h;
        int new_scan_line,scan_line;

        image_type_t fng_camera;
        HAND_TYPE hand;
        ROLL_DIRECTION capt_dir;


        /*
        ** parameters controlling the states of the rolling
        */
        int edge_tol = 20;
        int active_threshold = 150;
        int enough = 10;
        int extra = 25;
        int max_columns_in_step = 50;
        enum
        {
                BLANK,
                PRESS,
                ROLL_RIGHT,
                ROLL_LEFT,
                LIFT,
                DONE
        ) capt_state;

        /*
        ** check what camera to use     and required direction
        */

        if (finger_type_check(fng_num, &fng_camera, &hand, &capt_dir)
            || fng_camera != camera)
        return(CAPT_WRONG_TYPE);

        my_aia_init((UBYTE *)img_store_addr);

        aia_size(&gvram,&s_w,&s_h);

        /*
        ** image offset should be matched in image_capture() and display_camera()
```

```
**  these pointers are only used for the plain
*/
src_start_ptr = (UBYTE *)GRB_VRAM_END - SCR_W * 24 + 79;
dst_start_ptr = (UBYTE *)DPY_VRAM_BASE + SCR_W * 4 + 12;

if( camera == PLAIN )
{
        src_start_ptr += 20;
        dst_start_ptr += SCR_W * (732 - IMG_PLAIN_H);
}


/*
**  Get attributes of image and put into structure( IMAGE_ATTR_T )
*/
image.camera = camera;
image.hand = hand;
image.store_addr = img_store_addr;
image.src_addr = src_start_ptr;
image.dst_addr = dst_start_ptr;
image.fng_num = fng_num;

CaptureScreen( image, fng_num, nemap );


/*
**  Load the Equ data that were created during calibration for ROLL
**  or PLAIN from the Dram to EquDram, when the current calling camera
**  is different from the previous camera.
*/
if ( equref_camera != camera )
        load_equref_to_EquDram (camera);


/*
**  Wait for key pressed
*/
key_pressed = TYPE_NO_RESPONSE;
while( key_pressed != TYPE_SAVE_KEY )
{
        /*
        ** 1. Copy the straight thru function from Dram to MinSram
        ** 2. Turn grabber on
        */
        PrevInitAct( image );


        /*
        ** Preview operation
        */
        key_pressed = TYPE_NO_RESPONSE;
        while( key_pressed != TYPE_SCAN_KEY )
        {
                /*
                ** Display image by copy fingerprint from grabber VRAM to
                ** display VRAM.
                */
                display_fingerprint (camera, src_start_ptr, dst_start_ptr);

                key_pressed = OSBPend_key();

                if( (*prevscr_key_scr[key_pressed]) != NULL )
                        (* prevscr_key_scr[key_pressed] )( image, DRAW );
```

- 34 -

```
if( (*prevscr_key_act(key_pressed)) != NULL )
{
        ret = (*prevscr_key_act(key_pressed))( image );
        if( ret == CAPT_MISSFNG ){
                return( ret );
        }
}
)  /* end of while */


/*
** Capture operation
*/
if (capt_dir == RIGHT_DIR || capt_dir == LEFT_DIR)
{
        turn_grab_on_normal (image.camera, 1);

        capt_state=BLANK;

        key_pressed = TYPE_NO_RESPONSE;
        while( key_pressed != TYPE_SCAN_KEY && key_pressed != TYPE_SAVE_KEY)
        {
                switch (capt_state)
                {
                        case BLANK:
                                /*
                                **      wait for the finger to come down
                                **      display the active area when it is sensed
                                */
                                if (fz_detect_edges(&central_rows,edge_tol,active_threshold,
                                        &r_edge,&l_edge))
                                {
                                        capt_state=PRESS;
                                        far_r_edge=r_edge;
                                        far_l_edge=l_edge;
                                }
                                break;

                        case PRESS:
                                /*
                                **      display the active area
                                ** until both edges begin to roll in the correct direction
                                ** then save the back side of the active area
                                ** Allow the roll to go in either direction
                                */
                                if (fz_detect_edges(&central_rows,edge_tol,active_threshold,
                                        &r_edge,&l_edge))
                                {
                                        if (l_edge > far_l_edge+enough)
                                        {
                                                scan_line = smooth_scan_line(-1,r_edge,l_edge,capt_dir,
                                                        max_columns_in_step);
                                                /*
                                                ** Save the left side of the active area
                                                ** Don't update the display
                                                */
                                                save_columns(&gvram,&data_store,0,scan_line);
                                                capt_state=ROLL_RIGHT;
                                                break;
                                        }
```

- 35 -

```
If (r_edge < far_r_edge-enough)
{
        scan_line = smooth_scan_line(-1,r_edge,l_edge,capt_clr,
                max_columns_in_step);
        /*
        ** Save the left side of the active area
        ** Don't update the display
        */
        save_columns(&gvram,&data_store,scan_line,s_w);
        capt_state=ROLL_LEFT;
        break;
}
/*
** While not rolling just display the active columns
**
*/
if (l_edge-extra >= 0)
{
        if (r_edge+extra < s_w)
                display_columns_4_3(&gvram,&disp_window,l_edge-extra,
                        r_edge-l_edge+2*extra);
        else
                display_columns_4_3(&gvram,&disp_window,l_edge-extra,
                        s_w-l_edge+extra);
}
else if (r_edge+extra < s_w)
        display_columns_4_3(&gvram,&disp_window,0,r_edge+extra);
else
        display_columns_4_3(&gvram,&disp_window,0,s_w);


if (l_edge < far_l_edge)
        far_l_edge = l_edge;
if (r_edge > far_r_edge)
        far_r_edge=r_edge;
}
else
{

        min_move(&gvram,&data_store);
        display_columns_4_3(&data_store,&disp_window,0,s_w);
        capt_state=LIFT;

}
break;

case ROLL_RIGHT:
        /*
        ** save the area incremented by the scan line
        ** display right side of the active area
        **     until the right edge stops
        ** then save the right side of the active area
        */
        if (fz_detect_edges(&central_row,edge_tol,active_threshold,
                &r_edge,&l_edge))
        {
                new_scan_line = smooth_scan_line(scan_line,r_edge,l_edge,RIGHT_DIR,30);
                if (new_scan_line > scan_line)
                {
                        save_columns(&gvram,&data_store,scan_line,new_scan_line);
                        scan_line=new_scan_line;
                        if (r_edge+extra < s_w)
```

- 36 -

```
                              display_columns_6_3(&gvram,&disp_window,scan_line,
                                     r_edge+extra-scan_line);
                  else
                              display_columns_6_3(&gvram,&disp_window,scan_line,
                                     e_u-scan_line);
            /* display the scan line for debug
                  fill_col_6_3(&disp_window,scan_line,0);
            */
            )

            if (r_edge < far_r_edge - enough)
            {
                  save_columns(&gvram,&data_store,scan_line,e_u);
                  display_columns_6_3(&data_store,&disp_window,scan_line,
                        e_u-scan_line);
                  capt_state=LIFT;
            )
            if (l_edge < far_l_edge)
                  far_l_edge = l_edge;
            if (r_edge > far_r_edge)
                  far_r_edge=r_edge;


      )
      else
      (
            save_columns(&gvram,&data_store,scan_line,e_u);
            display_columns_6_3(&data_store,&disp_window,scan_line,e_u-scan_line);
            capt_state=LIFT;

      )

      break;

case ROLL_LEFT:
      /*
      ** save the area incremented by the scan line
      ** display left side of the active area
      **    until the left edge stops
      ** then save the left side of the active area
      */
      if (fz_detect_edges(&central_rows,edge_tol,active_threshold,
            &r_edge,&l_edge))
            (
                  new_scan_line = smooth_scan_line(scan_line,r_edge,l_edge,LEFT_DIR,30);
                  if (new_scan_line < scan_line)
                  (
                        save_columns(&gvram,&data_store,new_scan_line,scan_line);
                        scan_line=new_scan_line;
                        if (l_edge-extra >= 0)
                              display_columns_6_3(&gvram,&disp_window,l_edge-extra,
                                     scan_line-l_edge+extra);
                        else
                              display_columns_6_3(&gvram,&disp_window,0,scan_line);
                  /* display the scan line for debug
                        fill_col_6_3(&disp_window,scan_line,0);
                  */
                  )
                  if (l_edge > far_l_edge + enough)
                  (
                        save_columns(&gvram,&data_store,0,scan_line);
                        display_columns_6_3(&data_store,&disp_window,0,scan_line);
```

```
    turn_illuminator_on ();
    ptr = (ULONG *)PC_PARAMETERS;

    rollequ_exists = *ptr++;
    plainequ_exists = *ptr++;
    /* this is where version 1.3 loader puts it */
        palmequ_exists = *ptr++;
        palmwrp_exists = *ptr++;

    pers_exists = *ptr++;

    if ( pers_exists )
        {
        if (*ptr != 255) recursive_factor = *ptr;
        ptr++;
        if (*ptr != 255) fngr_desired_equ_value = *ptr;
        ptr++;
        if (*ptr != 255) changing_threshold = *ptr;
        ptr++;
        if (*ptr != 255) too_dark_value = *ptr;
        ptr++;
        if (*ptr != 255) roll_offset = *ptr;
        ptr++;
        if (*ptr != 255) plain_offset = *ptr;
        ptr++;
        if (*ptr != 255) palm_offset = *ptr;
        ptr++;
        if (*ptr != 255) palm_desired_equ_value = *ptr;
        ptr++;
        }

        /* this is where version 1.4 and beyond loader keeps it
    palmequ_exists = *ptr++;
        palmwrp_exists = *ptr++;
        */

#ifndef PALM_SCANNER
    init_asp_grabber ();

    calculate_minfuncLUT_to_Dram ();
    calculate_straight_thruLUT_to_Dram ();
    calculate_togglingLUT_to_Dram ();
#else
    /*
    ** Turn off grabber
    */
    *(UBYTE *)GRABBER_CTRL0_BASE = 0x0;
    *(UBYTE *)GRABBER_CTRL0_BASE = 0x0;

    load_straight_thruLUT_to_MinSram();
#endif
) /* initialize_34020 */

#ifndef PALM_SCANNER

/* ....................................................................*/
/*
** copy the LUTTBL from the given source address (in Dram by BYTE pointer)
** to MinSram (by LONG pointer)
*/
```

```c
image_type_t equref_camera = NON_CAMERA;    /* which camera equ in EquDram */
int rollequ_exists = 0;          /* 1=ROLL equ exists */
int plainequ_exists = 0;         /* 1=PLAIN equ exists */
int palmequ_exists = 0;          /* 1=PALM equ exists */
int palmwrp_exists = 0;          /* 1=PALM dewarping exists */


/* ------------------- Local Variable definition -------------------*/
/* this sets up the defaults if no parameter file exists */

static int para_exists = 0;                /* 1=image para file exists */
static ULONG recursive_factor = 66;/* sensitivity of capture to field rate */
static ULONG fngr_desired_equ_value = 252;/*value of blank platen equalized*/
static ULONG desired_equ_value = 252;  /* value of blank platen equalized */
static ULONG changing_threshold = 13;  /* percentage change of final to post
                     scan image, if not exceeded will white/tag out the data */
static ULONG too_dark_value = 5;        /* to remove A/D offset */
static ULONG roll_offset = 10;   /* ROLL peak histogram value */
static ULONG plain_offset = 10;  /* PLAIN peak histogram value */
static ULONG palm_offset = 10;   /* PALM peak histogram value */
static ULONG palm_desired_equ_value = 252;/*value of blank platen equalized*/
static ULONG detect_level = 150;  /* detection level for active area */


/* ------------------- External functions -------------------*/
extern VOID init_ramdac (VOID);
extern VOID init_asp_grabber (VOID);
extern VOID send_command_to_BIC (UBYTE command_to_send);
extern VOID turn_illuminator_on (VOID);



/* ------------------- Function Prototypes -------------------*/


VOID copy_image_to_Dram (image_type_t camera, ULONG img_store_addr,
     int ht_bytes, int wd_words);
VOID initialize_34020 (VOID);
VOID load_equref_to_EquDram (image_type_t camera);
VOID load_LUTTBL_to_MinDram (UBYTE *src_ptr);
VOID load_straight_thruLUT_to_EquDram (VOID);
VOID setup_graphic (VOID);

static VOID calculate_minfuncLUT_to_Dram (VOID);
static VOID calculate_straight_thruLUT_to_Dram (VOID);
static VOID calculate_taggingLUT_to_Dram (VOID);
static ULONG floor0 (ULONG a, ULONG b);
static VOID load_equLUT_to_EquDram (ULONG offset, ULONG dark_offset, ULONG desired);
static VOID load_straight_thruLUT_to_MinDram (VOID);



/* --------------------------------------------------------------*/
/*
** Initialize the Ramdac, the GSP registers, setup the graphic functions
** and create all LUTs.
*/
VOID initialize_34020 (VOID)
{
    volatile ULONG *ptr;

    *((USHORT *)DPYCTL) = 0;
    asm ("        NWAIT        ");
    init_ramdac ();
    setup_graphic ();
```

```
/*
** Filename: gsp_init.c
** Purpose:  GSP Initialization.
** Date:     08/10/93
** Author:   Joyce Young
** History:
**  02/18/94 Joyce Young, modify the offset of PLAIN active image
**  05/05/94 Joyce Young, if image parameter is 255, set default value
**  06/03/94 Joyce Young, adjust the offset of ROLL/PLAIN active image in
**           WINDRAM for new board
**  06/28/94 Joyce Young, turn_illuminator_on in initialize_34020, otherwise,
**           turning illuminator off during calibration and then receiving
**           ABORT command will cause the illuminator off after ABORT done
**  07/14/94 Joyce Young, change default roll_peak_illum_level = 10 &
**           plain_peak_illum_level = 10
**  08/11/94 TS, fix up offsets so that they add to the offsets calculated
**           from the dark level histogram instead of replacing it.
**  09/05/94 TS, change tagging so that a 0 change_threshold value causes
**           everything to be tagged.
**  09/22/94 JY, move setup of talon_font(CORPUS29) & talon_font(CORPUS49)
**           from PrintTitle() to setup_graphic(), otherwise, if calling
**           PrintTitle() several times, will mess up the talon_font table
**  09/29/94 TS, fixed error in Equ Sram initialization - needed to subtract
**           out offset after scaling. Cause severe clipping at white.
**  01/18/95 JY, read existing of palm.equ from PC_PARAMETERS
**  01/20/95 JY, delete unnecessary include files
**  01/23/95 JY, load palm.equ to EQU_DRAM if file exists
*/


/* .............................. Includes .............................. */
#include <gspreg.h>
#include <gsptypes.h>
#include <gspglobs.h>

#include "stdtypes.h"
#include "gsp_font.h"
#include "img_defs.h"
#include "mem_addr.h"
#include "mem_allc.h"
#ifndef PALM_SCANNER
#include "gsp_imgs.h"
#endif


/* .................... Extern Variable definition .................... */


/*
** variables for the graphic
*/
extern FONT corpus29, corpus49;


/* .................... Global Variable definition .................... */

FONTINFO fontinfo;
short smlfont_id, bigfont_id;
short smlfont_charhigh, bigfont_charhigh;
short smlfont_charwide, bigfont_charwide;


/*
** variables for the image processing
*/
```

```
                else
                        new_scan_line = old_scan_line;
        )
        else if (roll_dir == LEFT_DIR)
        (
                new_median = (scan_split*left + (8-scan_split)*right) /8;
                if (new_median < old_scan_line)
                (
                        new_scan_line = (factor*new_median + (8-factor)*old_scan_line + 4)/8;
                        if (new_scan_line < old_scan_line - max_columns)
                                new_scan_line = old_scan_line - max_columns;
                )
                else new_scan_line = old_scan_line;
        )
        else new_scan_line = new_median;

        return(new_scan_line);

) /* int smooth_scan_line */

/*******************************************************************
**
** replace any untagged pixels (even values) in the destination area
** with value
**
*/
void fill_even_pixels(MIN *dest,int value)
{
        register UBYTE *pp;
        register int vv,jj;

        int w,h,ii;

        vv = value;
        mim_size(dest,&w,&h);

        for (ii=0; ii<h; ii++)
        {
                pp = mim_adr(dest,0,ii);
                jj =w;
                do
                {
                        if (!(*pp & 1)) *pp = vv;
                        pp++;
                }
                while (--jj);
        }
)
/* ................................................................*/
/* end of fngrcapt.c */
```

```
                    mia_new(&src_rows,mia_adr(source,0,11),4*mia_inc(source),w,h/4);
                    mia_new(&dst_rows,mia_adr(dest,0,11),3*mia_inc(dest),0,0);
                    mia_move(&src_rows,&dst_rows);
            )
    ) /* display_4_3v */


    /* ...........................................................*/
    /*
    ** func name: VOID fill_col_4_3()
    ** Purpose:   fill a column in a range reduced by 4/3
    ** Returns:   None
    */
    VOID fill_col_4_3(MIM *dest, int col, int value)
    {
            register int ii,k,d_i;
            register unsigned char *d;
            int w,h;

            k = value;
            mia_size(dest,&w,&h);
            d = mia_adr(dest,col*3/4,0);
            d_i = mia_inc(dest);

            ii=h;
            do
            {
            *d=k;
            d+=d_i;
            ) while (--ii);

    ) /* fill_col_4_3 */


    /* ...........................................................*/
    /*
    ** func name: int smooth_scan_line()
    ** Purpose:   even out the scan line steps and limit them to a maximum size.
    ** Returns:   new scan line
    */
    int smooth_scan_line(int old_scan_line, int right, int left,
        ROLL_DIRECTION roll_dir, int max_columns)
    {
            int new_median, new_scan_line;
            int scan_split = 5;
            int factor = 4;

            if (old_scan_line < 0 )
            {
                    new_median = (right + left) / 2;
                    return (new_median);
            )

            if (roll_dir == RIGHT_DIR)
            {
                    new_median = (scan_split*right + (8-scan_split)*left) /8;
                    if (new_median > old_scan_line)
                    {
                            new_scan_line = (factor*new_median + (8-factor)*old_scan_line + 4)/8;
                            if (new_scan_line > old_scan_line + max_columns)
                                    new_scan_line = old_scan_line + max_columns;
                    )
```

```
        case STYPE_ROLL_LEFT_LITTLE:
                *camera = ROLL;
                *hand = LEFT;
                *desmear = LEFT_DIR;
                break;

        case STYPE_PLAIN_TWO_THUMBS:
                *camera = ROLL;
                *hand = RIGHT;
                *desmear = NO_DIR;
                return(1);
                break;

        case STYPE_PLAIN_RIGHT4:
                *camera = PLAIN;
                *hand = RIGHT;
                *desmear = NO_DIR;
                break;

        case STYPE_PLAIN_LEFT4:
                *camera = PLAIN;
                *hand = LEFT;
                *desmear = NO_DIR;
                break;

        case STYPE_PLAIN_RIGHT_THUMB:
                *camera = ROLL;
                *hand = RIGHT;
                *desmear = NO_DIR;
                break;

        case STYPE_PLAIN_LEFT_THUMB:
                *camera = ROLL;
                *hand = LEFT;
                *desmear = NO_DIR;
                break;

        default:
                return( 1 );
    }
    return (0);
} /* finger_type_check */

/* ...................................................................*/
/*
/*
** Func name: VOID display_4_3v()
** Purpose:   display the whole fingerprint with a 4:3 reduction in the
**            vertical direction, drop every fourth line
** Returns:   None
*/
VOID display_4_3v(MIN *source,MIN *dest)
{
        MIN src_rows,dst_rows;
        int w,h,s_1,ll;

        min_size(source,&w,&h);


        for (ll=0; ll<3; ll++)
        {
```

```
                return(0);
} /* fs_detect_edges */


/* ..............................................................*/
/*
** Func name: save_columns()
** Purpose:   block transfer selected columns of the source to the image
** Return:    None
*/
VOID save_columns(MIM *src,MIM *dest,int start, int stop)
{
        int s_w,s_h;
        MIM src_cols,dest_cols;

        mim_size(src,&s_w,&s_h);
        mim_subset(src,&src_cols,start,0,stop-start,s_h);
        mim_subset(dest,&dest_cols,start,0,0,0);
        mim_move(&src_cols,&dest_cols);
} /* save_columns */


/* ..............................................................*/
/*
** Func name: finger_type_check()
** Purpose:   decode the finger type, camera, whether to desmear and validity
** Returns:   0 - if no error
**            1 - otherwise
*/
int finger_type_check(scan_type_t frg_num,image_type_t *camera,
        HAND_TYPE *hand, ROLL_DIRECTION *desmear)
{
        /*
        ** decode the finger type, camera, whether to desmear and validity
        ** return (0) if no error
        */
        switch ( frg_num )
        {
                case STYPE_ROLL_RIGHT_THUMB:
                        *camera = ROLL;
                        *hand = RIGHT;
                        *desmear = LEFT_DIR;
                        break;

                case STYPE_ROLL_RIGHT_INDEX:
                case STYPE_ROLL_RIGHT_MIDDLE:
                case STYPE_ROLL_RIGHT_RING:
                case STYPE_ROLL_RIGHT_LITTLE:
                        *camera = ROLL;
                        *hand = RIGHT;
                        *desmear = RIGHT_DIR;
                        break;

                case STYPE_ROLL_LEFT_THUMB:
                        *camera = ROLL;
                        *hand = LEFT;
                        *desmear = RIGHT_DIR;
                        break;

                case STYPE_ROLL_LEFT_INDEX:
                case STYPE_ROLL_LEFT_MIDDLE:
                case STYPE_ROLL_LEFT_RING:
```

```
        return( CAPT_WRONG_TYPE );


        return( CAPT_SAVEING );
  /* CaptSaveAct */


/* .. .........................................................*/
**
** func name: int fx_detect_edges()
** Purpose:  Determine the right and left edges of the contact strip of
**           the fingerprint
** Returns:  0 - if there is no contact
**           1 - otherwise
*/
int fx_detect_edges(MIN *row, int edge_count, int threshold,
        int *right_edge, int *left_edge)
{
        int width,height,ii,sum;
        unsigned char *lp,*rp;


        /*
        **      Get the center row from the image
        **  Or together all the tag bits
        */
        mim_fill(&detect_row, 0);
        mim_move_c(row,&detect_row,0x200c);
        mim_size(row, &width, &height);

        lp = mim_adr(&detect_row,0,0);
        rp = lp+width-1;

        /* find the left edge by adding up and the right edge by adding down */
        sum = 0;
        for (ii=0; ii<width; ii++)
        {
                if (*lp++ & 1) {
                        sum++;
                        if (sum >= edge_count)
                                break;
                }
        }
        *left_edge=ii;

        sum = 0;
        for (ii=width-1; ii>0; ii--)
        {
                if (*rp-- & 1) {
                        sum++;
                        if (sum >= edge_count)
                                break;
                }
        }
        *right_edge=ii;

        /*
        **      there is contact if the right edge is farther right than the left
        **      edge
        */
        if (*right_edge > *left_edge)
                return(1);
        else
```

- 45 -

```
       */
       if( key_pressed == TYPE_SCAN_KEY )
                turn_grab_on (image.camera, 1);


       return( response_msstrg(key_pressed) );
) /* PrevSaveAct */


/* ..................................................................*/
/*
** func name: CaptScanAct()
** Purpose:   Actions when SCAN key pressed in capture screen.
** Returns:   CAPT_NO_RESPONSE
** History:   01-31-94 Joyce Young: Add return value
*/
capture_t CaptScanAct( IMAGE_ATTR_T image )
{
       /*--- Turn grabber off ---*/
       turn_grab_off (image.camera);
       return (CAPT_NO_RESPONSE);
}


/* ..................................................................*/
/*
** func name: CaptSaveAct()
** Purpose:   Actions when SAVE key pressed in capture screen.
** Returns:   CAPT_SAVEING/CAPT_WRONG_TYPE
** History:   02-18-94 Joyce Young: Pass camera in copy_image_to_Dram ()
*/
capture_t CaptSaveAct( IMAGE_ATTR_T image )
{
       int  wd_words, ht_bytes, w, h;


       /*
       **       grabber should already be off
       */


       if (image.camera == PLAIN)
       {
       /*--- Tag the images for changes from the remainder image ---*/
       load_LUTTBL_to_MinSram ((UBYTE *)FNGR_TAGGING_LUTTBL);
       turn_grab_on (image.camera, 1);
       turn_grab_off (image.camera);
       display_fingerprint (image.camera, image.src_addr, image.dst_addr);
                fill_even_pixels(&plain_disp_window,254);
                /*--- Download image to the GSP DRAM ---*/
                wd_words = PLAIN_NO_WORDS;
                ht_bytes = PLAIN_HT_BYTES;
       copy_image_to_Dram (image.camera, image.store_addr, ht_bytes, wd_words);
       }
       else if (image.camera == ROLL)
       {
                load_LUTTBL_to_MinSram ((UBYTE *)FNGR_TAGGING_LUTTBL);
                turn_grab_on_normal(image.camera,1);
                turn_grab_off(image.camera);
                min_move(&minram,&data_store);
                min_size(&data_store,&w,&h);
                display_columns_4_3(&data_store,&disp_window,0,w);
                fill_even_pixels(&disp_window,254);
       }
       else
```

```
/* ...............................................................*/
/*
** Func name: PrevScanAct()
** Purpose:  Actions when SCAN key pressed in preview screen.
** Returns:  CAPT_NO_RESPONSE/CAPT_WRONG_TYPE, no captured images or
**           wrong camera type.
*/
capture_t PrevScanAct( IMAGE_ATTR_T image )
{
        /*.... Turn grabber off ...*/
        turn_grab_off( image.camera );


        /*
        ** Load minfunc from Drem to MinSrem
        */
#if IMG_DEBUG
        load_straight_thruLUT_to_EqdSrem ();
        load_LUTTBL_to_MinSrem ((UBYTE *)STRAIGHT_THRU_LUTTBL);
#else
        load_LUTTBL_to_MinSrem ((UBYTE *)MINFUNC_LUTTBL);
#endif

   return( CAPT_NO_RESPONSE );
) /* PrevScanAct */

/* ...............................................................*/
/*
** Func name: PrevSaveAct()
** Purpose:  Actions when SAVE key pressed in preview screen.
** Returns:  CAPT_NO_RESPONSE/CAPT_MISSFNG, no captured images or missing
**           finger
** History:  01-31-96 Joyce Young: Turn on the grabber if SCAN key pressed
*/
capture_t PrevSaveAct( IMAGE_ATTR_T image )
{
        keytype_t key_pressed;
        capture_t response_misfng[] =
        {
                CAPT_NO_RESPONSE,
                CAPT_NO_RESPONSE,
                CAPT_NO_RESPONSE,
                CAPT_MISSFNG,
                CAPT_MISSFNG,
        );

        /*.... Turn grabber off ...*/
        turn_grab_off( image.camera );

        key_pressed = TYPE_NO_RESPONSE;
        while( key_pressed != TYPE_SCAN_KEY && key_pressed != TYPE_SAVE_KEY )
                key_pressed = OSOPend_key();

        if( (*misfngscr_key_scr[key_pressed]) != NULL )
                (* misfngscr_key_scr[key_pressed] )( image, DRAW );


        /*
        ** SCAN key will go to preview screen, as we turn off the grabber above,
        ** so the grabber needs to turn on here, otherwise, the finger image will
        ** not show on the preview screen
```

```
        if( camera == ROLL )
                remdec_display_cross_symbol();

        src_start_ptr = (UBYTE *)GRB_VRAM_END;
        src_start_ptr -= SCR_V * 30 - 79;
        dst_start_ptr = (UBYTE *)DPY_VRAM_BASE;
        dst_start_ptr += SCR_V * 4 + 12;

        if ( camera == PLAIN )
        {
                src_start_ptr += 20;
                dst_start_ptr += SCR_V * (732 - IMG_PLAIN_H);
        }

        load_straight_thruLUT_to_EqSram ();
        load_LUTTBL_to_MinSram ((UBYTE *)STRAIGHT_THRU_LUTTBL);
        turn_grab_on (camera, 1);

        for (;;)
        {
                display_fingerprint (camera, src_start_ptr, dst_start_ptr);

                switch ( key_type = OSGPend_key () )
                {
                        case TYPE_SCAN_KEY:
                        case TYPE_SAVE_KEY:
                        case TYPE_ABORT:
                                turn_grab_off (camera);
                                if( camera == ROLL )
                                        remdec_clear_cross_symbol();
                                return;
                }
        }
} /* display_camera */

/* ..............................................................*/
/*
** Func name: PrevInitAct()
** Purpose:   Actions when enter a preview screen.
** Returns:   CAPT_NO_RESPONSE, no captured images
*/
capture_t PrevInitAct( IMAGE_ATTR_T image )
{
        /*
        ** load straight thru LUT from Dram to MinSram
        */
#if IMG_DEBUG
        load_straight_thruLUT_MEM_to_EqSram ();
#else
        load_LUTTBL_to_MinSram ((UBYTE *)STRAIGHT_THRU_LUTTBL);
#endif

        /*
        ** Turn grabber on 2 frame, otherwise, the second finger image will
        ** display garbage lines if using multiple finger buffers
        */
        turn_grab_on (image.camera, 2);

        return( CAPT_NO_RESPONSE );
} /* PrevInitAct */
```

```
                                        if( ret == CAPT_SAVEIMG )
                                        {
                                                return( ret );
                                        }
                                }
                        }  /* end of while */
                }        /* end of if */

        }  /* end of while */
} /* image_capture */

/* .............................................................*/
/*
** func name: VOID display_fingerprint()
** Purpose:  Draw the fingerprint in reduced form in display window.
** The fingerprint thru the TG's camera is stored upside-down in the GVRAM.
** the horizontal reduction is done in hardware
** the vertical reduction is done by dropping 1 of 4 lines in the roll
** and 1 of 2 lines in the plain.
*/
VOID display_fingerprint (image_type_t camera, UBYTE *src_start_ptr,
        UBYTE *dst_start_ptr)
{
        UBYTE *src_ptr, *dst_ptr;
        int i;
        if ( camera == ROLL )
        {

                /* set up 4/3 reduction */
                display_4_3v(&gvram_reduced,&disp_window);


        }
        else
        {
                /*
                ** Only display every other lines;
                ** so increment the source array 2 lines, dest. array 1 line
                ** The first display line is in the middle of the screen.
                */
                poke_breg (SADDR, src_start_ptr);       /* B0: starting addr of source array */
                poke_breg (DADDR, dst_start_ptr);       /* B2: starting addr of dest.  array */
                poke_breg (SPTCH, PLAIN_GRGVRAM_PITCH); /* B1: the pitch of source array */
                poke_breg (DPTCH, PLAIN_DPTVRAM_PITCH); /* B3: the pitch of dest. array */
                poke_breg (DYDX,  PLAIN_DYDX_VAL);       /* B7: DX: array width, DY: array height */
                asm ("        PIXBLT L, L ");
        }
} /* display_fingerprint */

/* .............................................................*/
/*
** func name: VOID display_camera()
** Purpose:  Display the Roll or Plain image from the given camera
**           (Roll or Plain) by using straight_thru function
** Returns:   None
*/
VOID display_camera (image_type_t camera)
{
        keytype_t  key_type;
        UBYTE *src_start_ptr, *dst_start_ptr;
```

– 49 –

```
                                        mim_move(&gvram,&data_store);
                                        break;

                            case ROLL_RIGHT:
                                        save_column(&gvram,&data_store,scan_line,e_w);
                                        break;

                            case ROLL_LEFT:
                                        save_column(&gvram,&data_store,0,scan_line);
                                        break;
                    }
                    turn_grab_off(camera);
                    mim_move(&data_store,&gminram);

            }

            if( (*captscr_key_scr[key_pressed]) != NULL )
                    (*captscr_key_scr[key_pressed])( image, DRAW );

            if( (*captscr_key_act[key_pressed]) != NULL )
            {
                    ret = (*captscr_key_act[key_pressed])( image );
                    if( ret == CAPT_SAVEING )
                    {
                            return( ret );
                    }
            }
    }   /* end of while */
}


else if ((camera == ROLL && capt_dir == NO_DIR) || camera == PLAIN)
{

        turn_grab_on (image.camera, 1);

        key_pressed = TYPE_NO_RESPONSE;
        while( key_pressed != TYPE_SCAN_KEY && key_pressed != TYPE_SAVE_KEY)
        {
                /*
                ** Display image by copy fingerprint from grabber VRAM to
                ** display VRAM.
                */
display_fingerprint (camera, src_start_ptr, dst_start_ptr);

                key_pressed = OSQPend_key();

                /*
                **      turn off the grabber when the save key is pressed
                */
                if(key_pressed == TYPE_SAVE_KEY)
                {
                        turn_grab_off(camera);
                }

                if( (*captscr_key_scr[key_pressed]) != NULL )
                        (*captscr_key_scr[key_pressed])( image, DRAW );

                if( (*captscr_key_act[key_pressed]) != NULL )
                {
                        ret = (*captscr_key_act[key_pressed])( image );
```

```
                    capt_state=LIFT;
            )

            if (l_edge < far_l_edge)
                    far_l_edge = l_edge;
            if (r_edge > far_r_edge)
                    far_r_edge=r_edge;


        )
        else
        (
                save_columns(&gvram,&data_store,0,scan_line);
                display_columns_4_3(&data_store,&disp_window,0,scan_line);
                capt_state=LIFT;

        )


        break;

    case LIFT:
            /*
            ** wait for the active area to go away
            ** when it does, detect the changed areas and
            ** tag the final image
            */
            if (fl_detect_edges(&central_row,edge_tol,active_threshold,
                    &r_edge,&l_edge))
            (
            )
            else
            (
                    turn_grab_off(image.camera);
                    capt_state=GONE;
            )
            break;

    case GONE:
            /*
            ** don't do anything
            */
            break;

    default:
            break;
)

key_pressed = OSQPend_key();

/*
**      fix up the data store in case the save key is pressed
** before the end of the roll
** Then turn the grabber off and move the data_store
** back into the grabber min dram.
*/
if(key_pressed == TYPE_SAVE_KEY)
(
        switch (capt_state)
        (
                case BLANK:
                case PRESS:
```

```
VOID load_LUTTBL_to_MinSram (UBYTE *src_ptr)
{
    register ULONG *d;
        register UBYTE *s;
        register LONG ii;

    s = src_ptr;
    d = (ULONG *)MIN_SRAM_BASE;
        ii = 65536;
        do
        *d++ = *s++;
        while (--ii);

} /* load_LUTTBL_to_MinSram */

/* .................................................................*/
/*
** Calculate the Min function and store to the Dram,
** later will copy to MinSram
*/
VOID calculate_minfuncLUT_to_Dram (VOID)
{
    UBYTE a_d, mem, *ptr, out;
    ULONG cond;

    ptr = (UBYTE *)MINFUNC_LUTTBL;

    for ( mem = 0; mem < 256; mem++ )
        for ( a_d = 0; a_d < 256; a_d++ )
        {
            cond = (UBYTE)floor0 (mem, a_d) * recursive_factor;

            if ( cond >= 50 )
                out = mem - (cond + 50) / 100;
            else if ( cond > 0 )
                out = (UBYTE)floor0 (mem, 1);
            else
                out = mem;

            if (a_d < detect_level)
                        *ptr++ = out | 1;
                    else
                            *ptr++ = out & 0xfe;
        }
} /* calculate_minfuncLUT_to_Dram */

/* .................................................................*/
/*
** Calculate the Straight Thru and store to the Dram,
** later will copy to MinSram
*/
VOID calculate_straight_thruLUT_to_Dram (VOID)
{
    UBYTE a_d, mem, *ptr;

    ptr = (UBYTE *)STRAIGHT_THRU_LUTTBL;

    for ( mem = 0; mem < 256; mem++ )
        for ( a_d = 0; a_d < 256; a_d++ )
                    if (a_d < 150)
```

```
                              *ptr++ = a_d | 1;
                         else
                              *ptr++ = a_d & 0xfe;

) /* calculate_straight_thruLUT_to_Dram */

/* ...........................................................*/
/*
** Calculate the Tagging function and store to the Dram,
** later will copy to MinDram
*/
VOID calculate_taggingLUT_to_Dram (VOID)
{
    UBYTE a_d, mem, *ptr;
    UBYTE vv;

    ptr = (UBYTE *)FNGR_TAGGING_LUTTBL;

    for ( mem = 0; mem < 256; mem++ )
        for ( a_d = 0; a_d < 256; a_d++ )
        {
            if (changing_threshold > 0)
            {
                vv = ((ULONG)a_d * (ULONG)(100 - changing_threshold) / (ULONG)100);

                if ( (vv < mem) || (a_d < too_dark_value) )
                    *ptr++ = mem & 0xFFFFFFFE;    /* untag (white) */
                else
                    *ptr++ = mem | 1;             /* tag (dark) */
            }
            else
                *ptr++ = mem | 1;
        }
) /* calculate_taggingLUT_to_Dram */

/* ...........................................................*/
/*
** Copy image from MinDram (2048 * 1024) to Dram (ROLL:976, 976... or
** PLAIN:1600, 1600... continuously).
** The image is upside down in MinDram and is stored to Dram rightside up
*/
VOID copy_image_to_Dram (image_type_t camera, ULONG img_store_addr,
    int ht_bytes, int wd_words)
{
    int ii, jj;
    ULONG *src_ptr, *dst_ptr;

    /*
    ** calculate the offset of the active image
    ** NOTE: image offset in copy_image_to_Dram(), load_eqaref_to_EqDram(),
    **       process_calibration() should be the same.
    */
    dst_ptr = (ULONG *)img_store_addr;
    src_ptr = (ULONG *)MIN_DRAM_BASE;

    if ( camera == ROLL )
        {
        src_ptr += MINDRAM_WD_WORDS * (ht_bytes + MINDRAM_ROLL_ROW_B_OFFSET)
            + MINDRAM_ROLL_COL_W_OFFSET;     /* 512 * (960 + 38) + 24 */
        }
```

```
else
    (
    src_ptr += MINDRAM_WD_WORDS * (ht_bytes + MINDRAM_PLAIN_ROW_B_OFFSET)
        + MINDRAM_PLAIN_COL_V_OFFSET;    /* 512 * (9.'6 + 31) + 39 */
    )

.for ( ii = 0; ii < ht_bytes; ii++ )
    (
    for ( jj = 0; jj < wd_words; jj++ )
        *dat_ptr++ = *src_ptr++;


    /*
    ** go to the beginning of the previous row
    */
    src_ptr -= MINDRAM_WD_WORDS + wd_words;

    )
) /* copy_image_to_Dram */

/* ...........................................................*/
/*
** copy the equalize reference of the given camera from the Dram to EqDram
*/
VOID load_equref_to_EqDram (image_type_t camera)
(
    ULONG *src_ptr, *dat_ptr;
    ULONG offset_level;
    int wd_words, ht_bytes;
    int ii, jj, equ_exists;
    ULONG dark_offset = 0;
        register ULONG *s,*d;

    if ( camera == ROLL )
        (
        ht_bytes = ROLL_HT_BYTES;
        wd_words = ROLL_WD_WORDS;
        src_ptr = (ULONG *)ROLL_EQU_REFTBL;
        equ_exists = rollequ_exists;
        offset_level = roll_offset;
        )
    else
        (
        ht_bytes = PLAIN_HT_BYTES;
        wd_words = PLAIN_WD_WORDS;
        src_ptr = (ULONG *)PLAIN_EQU_REFTBL;
        equ_exists = plainequ_exists;
        offset_level = plain_offset;
        )

    if ( equ_exists )
        (
        /*
        ** if the equalization reference exists we get the value of the
        ** dark offset calculated from the peak level of
        ** the dark image histogram.
        */
        dark_offset = *(UBYTE *)src_ptr;


        /*
        ** The image at the EqDram should be exactly as MinDram. So use all
        ** MinDram defines.
```

```
** calculate the offset of the active image
** NOTE: image offset in copy_image_to_Dram(),
**       load_equref_to_EqDram(), process_calibration() should be
**       the same.
*/
dat_ptr = (ULONG *)EQU_DRAM_BASE;

if ( camera == ROLL )
    {
    dat_ptr += MINDRAM_WD_WORDS * (ht_bytes + MINDRAM_ROLL_ROW_B_OFFSET)
        + MINDRAM_ROLL_COL_V_OFFSET;    /* 512 * (960 + 38) + 26 */
    }
else
    {
    dat_ptr += MINDRAM_WD_WORDS * (ht_bytes + MINDRAM_PLAIN_ROW_B_OFFSET)
        + MINDRAM_PLAIN_COL_V_OFFSET;   /* 512 * (976 + 31) + 39 */
    }


        /*
** transfer the bytes to the words
*/
        s = src_ptr;
        i=ht_bytes;
    do
    {
            d = dat_ptr;
            jj=wd_words;
            do
            {
                *d++ = *s++;
            } while (--jj);
        dat_ptr += MINDRAM_WD_WORDS;

        } while (--ii);

    #if INO_DEBUG
    load_straight_thruLUT_MEM_to_EquSram ();
    #else
    load_equLUT_to_EQUSram (offset_level,dark_offset,desired_equ_value);
    #endif
    }
else
    {
    load_straight_thruLUT_to_EquSram ();
    }

  equref_camera = camera;
} /* load_equref_to_EqDram */

#else /* PALM_SCANNER */

/* .............................................................*/
/*
** Calculate the straight thru and store to the MinSram for PALM scanner
*/
VOID load_straight_thruLUT_to_MinSram (VOID)
{
    ULONG s_d, mem, *ptr;

    ptr = (ULONG *)MIN_SRAM_BASE;
```

```
    for ( mem = 0; mem < 256; mem++ )
         for ( e_d = 0; e_d < 256; e_d++ )
               {
               *ptr++ = e_d | 1;
               }
) /* load_straight_thruLUT_to_MinSram */

/* ................................................................*/
/*
** copy the equalize reference of PALM from the Dram to EquDram
*/
VOID load_palm_equref_to_EquDram (VOID)
{
    UBYTE *src_ptr;
    ULONG *dst_ptr;
    int ii;
    ULONG dark_offset;

    if ( palmequ_exists )
        {
        src_ptr = (UBYTE *)PALM_EQU_REFTBL;
        dst_ptr = (ULONG *)EQU_DRAM_BASE;

        /*
        ** we get the value of the dark offset calculated from the peak
        ** level of the dark image histogram.
        */
        dark_offset = *(ULONG *)src_ptr;

        src_ptr += 7;

        for ( ii = 0; ii < PALM_NO_PIXELS; ii++ )
            *dst_ptr++ = *src_ptr++;

#if IMG_DEBUG
        load_straight_thruLUT_MEM_to_EquSram ();
#else
        load_equLUT_to_EQUSram (palm_offset,dark_offset,desired_equ_value);
#endif
        }
     else
        load_straight_thruLUT_to_EquSram ();

     equref_camera = PALM_SCAN;
) /* load_equref_to_EquDram */

#endif    /* PALM_SCANNER */

/* .
** below functions are common for finger scanner and palm scanner
*/

/* ................................................................*/
/*
** Calculate the equalize LUT TABLE and store to the EQU_SRAM
*/
VOID load_equLUT_to_EQUSram (ULONG offset, ULONG dark_offset, ULONG desired)
{
    ULONG e_d, mem, *ptr, out, vv, ee;
```

- 56 -

```
ptr = (ULONG *)EQU_SRAM_BASE;

for ( a_d = 0; a_d < 256; a_d++ )
    for ( mem = 0; mem < 256; mem++ )
    {
        ee = floor0 (mem, dark_offset);

        if ( ee != 0 )
        {
            out = (ULONG)(desired+offset) *
                floor0(a_d,dark_offset) / ee;
            if (out > offset)
                out = out - offset;
            else out = 0;
        }
          else
            out = 254;

        if ( out < 0 )
            out = 0;

        if ( out > 254 )
            out = 254;

        *ptr++ = out | 1;
    }
) /* load_equLUT_to_EQUSram */

/* .................................................................*/
/*
** Calculate the Straight Thru of new image and store to the Equ Sram
*/
VOID load_straight_thruLUT_to_EquSram (VOID)
{
    ULONG a_d, mem, *ptr;

    ptr = (ULONG *)EQU_SRAM_BASE;

    for ( a_d = 0; a_d < 256; a_d++ )
        for ( mem = 0; mem < 256; mem++ )
            {
            *ptr++ = a_d | 1;
            }
) /* load_straight_thruLUT_to_EquSram */

#if IMG_DEBUG
/* .................................................................*/
/* .
** Calculate the Straight Thru of memory's image and store to the Equ Sram
** NOTE: This routine is for debugging purpose
*/
VOID load_straight_thruLUT_MEM_to_EquSram (VOID)
{
    ULONG a_d, mem, *ptr;

    ptr = (ULONG *)EQU_SRAM_BASE;

    for ( a_d = 0; a_d < 256; a_d++ )
        for ( mem = 0; mem < 256; mem++ )
```

```
            (
            *ptr++ = mem | 1;
            )
) /* load_straight_thruLUT_MEM_to_EqURam */
#endif

/* ...............................................................*/
/*
** Calculate the division to get rid of the overflow
*/
ULONG floor0 (ULONG a, ULONG b)
{
    if ( a >= b )
        return (a - b);
    else
        return (0);
} /* floor0 */

/* ...............................................................*/
/*
** Setup the basic graphic function
*/
void setup_graphic (void)
{
    set_config (0, 10);
    clear_whole_screen();

    smlfont_id = install_font (&corpus29);
    bigfont_id = install_font (&corpus49);

    talon_font[CORPUS29] = install_font (&corpus29);
    talon_font[CORPUS49] = install_font (&corpus49);

    select_font (smlfont_id);
    get_fontinfo (smlfont_id, &fontinfo);
    smlfont_charhigh = fontinfo.charhigh;
    smlfont_charwide = fontinfo.charwide;

    select_font (bigfont_id);
    get_fontinfo (bigfont_id, &fontinfo);
    bigfont_charhigh = fontinfo.charhigh;
    bigfont_charwide = fontinfo.charwide;
} /* setup_graphic */


/* end of gsp_init.c */
```

- 58 -

```
/* ...........................................................
 *
 * File name: userintf.c
 *
 * Purpose:   User interface on scanner for talon 1000.
 *
 * Date:      Jan-05-96
 *
 * Author:    Ellen Yu
 *
 * History:
 *   05-12-94 Joyce Young, fill WinDram with 0xFFFFFFFF is not needed.
 *   06-13-94 Joyce Young, add #ifndef NPA for NPA
 *   06-30-94 Ellen Yu,    turn grabber on 2 frame in PrevinitAct
 *   09-12-94 Joyce Young, add display_text() to display general message to
 *            call text_out() & kanji_out() separately
 *   09/22/94 JY, move setup of talon_font[CORPUS29] & talon_font[CORPUS49]
 *            from PrintTitle() to setup_graphic(), otherwise, if calling
 *            PrintTitle() several times, will mess up the talon_font table
 *   11-15-94 JY, delete #ifndef NPA, so keep displaying WINfunc image
 *   11-17-94 JY, add text_length() to get text length to call text_width()
 *            & kanji_length() separately
 *   11/18/94 JY, delete CAPT_ABORT case
 *   12-21-94 EY, fixed displaying WINfunc image
 *   01/21/95 JY, delete unnecessary include files
 *   02/17/95 TS, moved out button action routines to fngrcapt
 * ........................................................... */


/* ..................... Includes ....................... */
#include "stdtypes.h"
#include "colordef.h"
#include "coorddef.h"
#include "gsp_defs.h"
#include "gsp_font.h"
#include "gsp_text.h"
#include "imgglobs.h"
#include "mem_allc.h"


/* ................... External Functions ................ */

/* ................... Prototype ........................ */
VOID PrintTitle (VOID);
VOID display_text (SHORT xx, SHORT yy, VOID *string, SHORT knj_font);
SHORT text_length (VOID *string, SHORT knj_font);


/* ................... Global Vars ...................... */
SCRTEXT_T titletext[] =
{
   { CORPUS49, GREY6, WHITE, TITLE_TXT_X,    TITLE_TXT_Y,    TITLE_TXT_X,
     TITLE_FONT_N,    0, TITLE_TXT ),
   { CORPUS49, GREY4, WHITE, COPYRIGHT_TXT_X, COPYRIGHT_TXT_Y, COPYRIGHT_TXT_X,
     COPYRIGHT_FONT_N, 0, COPYRIGHT_TXT ),
#ifndef PALM_SCANNER
   { CORPUS29, GREY6, WHITE, VERSION_TXT_X,   VERSION_TXT_Y,   VERSION_TXT_X,
     VERSION_FONT_N,   0, FNGR_VER_TXT )
#else
   { CORPUS29, GREY6, WHITE, VERSION_TXT_X,   VERSION_TXT_Y,   VERSION_TXT_X,
```

```
          VERSION_FONT_N,     0, PALM_VER_TXT )
#endif
);


/* ................................................................ */



/* ................................................................ */
/*
** Display the given string on the (xx, yy) position for either English or
** Kanji
*/
VOID display_text (SHORT xx, SHORT yy, VOID *string, SHORT knj_font)
{
#ifndef KANJI
   text_out (xx, yy, (char *)string);
#else
   kanji_out (xx, yy, (SHORT *)string, knj_font);
#endif
} /* display_text */


/* ................................................................ */
/*
** Get the length of the given string either for English or for Kanji
*/
SHORT text_length (VOID *string, SHORT knj_font)
{
   short txt_w;

#ifndef KANJI
   txt_w = text_width ((char *)string);
#else
   txt_w = kanji_length ((SHORT *)string, knj_font);
#endif

   return (txt_w);
} /* text_length */

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 *
 *  func name: PrintTitle()
 *
 *  Purpose:   Display the TALON 1000 title and copyright
 *
 *  Date:      Jan-05-96
 *
 *  Author:    Ellen Yu
 *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
VOID PrintTitle (VOID)
{
   SCRTEXT_T *txtptr;
   short txt_w;

   /*
   ** Company title
   */
   txtptr = &titletext[0];
   set_colors( txtptr->fcolor, txtptr->bcolor );
```

```
select_font( talon_font(txtptr->font) );

txt_w = text_width ( txtptr->strptr );
txtptr->xposn = (SCR_W - txt_w) / 2;
text_out( txtptr->xposn, txtptr->yposn, txtptr->strptr );


/*
** Copyright
*/
txtptr = &titletext[1];
set_colors( txtptr->fcolor, txtptr->bcolor );
select_font( talon_font(txtptr->font) );

txt_w = text_width ( txtptr->strptr );
txtptr->xposn = (SCR_W - txt_w) / 2;
text_out( txtptr->xposn, txtptr->yposn, txtptr->strptr );


/*
** Version and type of scanner
*/
txtptr = &titletext[2];
set_colors( txtptr->fcolor, txtptr->bcolor );
select_font( talon_font(txtptr->font) );

txt_w = text_width ( txtptr->strptr );
txtptr->xposn = (SCR_W - txt_w) / 2;
text_out( txtptr->xposn, txtptr->yposn, txtptr->strptr );

) /* PrintTitle */

/* end of userintf.c */
```

```
/* ............................................................... */
 *  File name: almio.c
 *  Purpose:   Routines for handling memory images in the 34010.
 *  Date:      Oct-05-93
 *  Author:    Tom Sartor
 *  History:
 * ............................................................... */


/* ....................... includes ....................... */
#include "almio.h"

void byteblt(),bytefll();

/* ............................................................... */
void mim_new(d,a,l,w,h)
MIM *d;
unsigned char *a;
int l,w,h;
{
    d->a = a;
    d->l = l;
    d->w = w;
    d->h = h;
}


/* ............................................................... */
void mim_subset(s,d,x,y,w,h)
MIM *s,*d;
int x,y,w,h;
{
    d->l = s->l;
    d->a = (s->a)+x+y*(d->l);
    d->w = w;
    d->h = h;
}


/* ............................................................... */
void mim_move(s,d)
MIM *s,*d;
{
    byteblt(s->a,(s->l)<<3,d->a,(d->l)<<3,s->h,s->w,0x000c);
}


/* ............................................................... */
void mim_move_c(s,d,c)
MIM *s,*d;
int c;
{
    byteblt(s->a,(s->l)<<3,d->a,(d->l)<<3,s->h,s->w,c);
}


/* ............................................................... */
unsigned char * mim_adr(s,x,y)
MIM *s;
int x,y;
{
    return((s->a)+x+y*(s->l));
}

/* ............................................................... */
```

- 62 -

```
void min_size(s, w, h)
MIN *s;
int *w,*h;
{
    *w = s->w;
    *h = s->h;
}


/* ............................................................ */
int min_inc(s)
MIN *s;
{
        return(s->i);
}


/* ............................................................ */
void min_fill(d, v)
MIN *d;
int v;
{
    bytefill(v&0xff,d->a,(d->i)<<3,d->h,d->w,0x000c);
}


/* ............................................................ */
void min_fill_c(d, v, c)
MIN *d;
int v,c;
{
    bytefill(v&0xff,d->a,(d->i)<<3,d->h,d->w,c);
}


/* ............................................................ */
/* end of minio.c */
```

```
/*
** Filename: grabber.c
** Purpose:  Routines to handle the Grabber.
** Date:     11/16/93
** Author:   Joyce Young
** History:
** 05/05/94 Joyce Young, Add mask register for display (GRABBER_CTRL1_BASE)
** 01/04/95 Ellen Yu, Change identatin style.
** 04/13/95 te - added different turn_grab_on_normal, modified turn_off,
**               and moved prototypes to gsp_func.h
** 04/19/95 EY - Added comments.
*/


#include "stdtypes.h"
#include "img_defs.h"
#include "mem_addr.h"
#include "gsp_func.h"


/* ................................................................*/
/*
** Initialize the GSP Grabber: setup the Video Control and Grabber Control.
*/
VOID init_gsp_grabber (VOID)
{
/*
** Setup the Video Control:
**   read the System Resources (address is the same as Video Control),
**   If the value of the SVGA Sync Input detected (bit 6) is 0 (normal is 1),
**   then set Video Source as External Sync Source (bit1-0 = 01), otherwise,
**   set as Internal Sync Source (bit1-0 = 00).
**
*/
    *(ULONG *)VIDEO_CONTROL_BASE = 0x00000000;
    *(ULONG *)GRABBER_CTRL1_BASE = 0xFFFFFFFF;

    /*
    ** Turn the Grabber Off
    */
    turn_grab_off (ROLL);
} /* init_gsp_grabber */


/* ................................................................*/
/*
** Turn Grab off, polling Grabber status till off (bit0 = 0).
*/
VOID turn_grab_off (image_type_t camera)
{
#ifndef DISABLE_GRAB
    /*.... Be sure bit 7 be cleared, no image reduction ....*/
    if ( camera == ROLL )
        *(UBYTE *)GRABBER_CTRL0_BASE = 0x70;
    else
        *(UBYTE *)GRABBER_CTRL0_BASE = 0xE0;

    /*
    ** wait for Grab is disabled (BIT0 becomes 1)
    */
    while ( !((*(UBYTE *)GRABBER_CTRL0_BASE) & 1) )
        ;
```

- 64 -

```
/*
** wait for the VBLANK start
** NOTE: need to wait VBLANK off/on, otherwise, Grabber sometimes does not
**   really turn off
*/
  while ( !((*(UBYTE *)GRABBER_CTRL0_BASE) & 2) )
      ;

  while ( ((*(UBYTE *)GRABBER_CTRL0_BASE) & 2) )
      ;
#endif
) /* turn_grab_off */

/* ..............................................................*/
/*
** Turn Grabber on with reduced display
** polling Grabber status till on (bit0 = 1).
*/
VOID turn_grab_on (image_type_t camera, int frame_num)
{
#ifndef DISABLE_GRAB
    int ii;

    if ( camera == ROLL )
        *(UBYTE *)GRABBER_CTRL0_BASE = 0xFF;
    else
        *(UBYTE *)GRABBER_CTRL0_BASE = 0xEF;


    /*
    ** wait for Grab is enabled (BIT0 becomes 0)
    */
    while ( ((*(UBYTE *)GRABBER_CTRL0_BASE) & 1) )
        ;

    for ( ii = 0; ii < frame_num; ii++ )
        {
        while ( !((*(UBYTE *)GRABBER_CTRL0_BASE) & 2) )
            ;

        while ( ((*(UBYTE *)GRABBER_CTRL0_BASE) & 2) )
            ;
        }
#endif
) /* turn_grab_on */

/* ..............................................................*/
/*
** Turn Grabber on with normal display
** polling Grabber status till on (bit0 = 1).
*/
VOID turn_grab_on_normal (image_type_t camera, int frame_num)
{
#ifndef DISABLE_GRAB
    int ii;

    if ( camera == ROLL )
        *(UBYTE *)GRABBER_CTRL0_BASE = 0x7F;
    else
        *(UBYTE *)GRABBER_CTRL0_BASE = 0xEF;
```

- 65 -

```
*/
typedef enum image_type_t
(
   ROLL,
   PLAIN,
   PALM_SCAN,
   NON_CAMERA
) image_type_t;

#endif    /* IMG_DEFS_H */
```

```
/*
** Filename: img_defs.h
** Purpose: Definitions to handle the images display or capture for Host,
**          DSP and GSP.
** Date:    10/11/93
** Author:  Joyce Young
** Revised:
** 11/02/93 - Ellen Yu
** 01/28/94 - Joyce Young, Add ROLL_RESOLUTION & PLAIN_RESOLUTION
** 02/02/94 - Ellen Yu, Seperate into img_defs.h and imgglobs.h.
** 02/02/94 - Joyce Young, Add the definitions for the PALM camera
** 11/21/94 - JY, Add #ifdef NEW_PALM to support new PALM
** 01/13/95 - JY, change palm pixels for new NPA palm
*/


#ifndef IMG_DEFS_H
#define IMG_DEFS_H


#define ROLL_WD_BYTES          960
#define ROLL_WD_WORDS          (ROLL_WD_BYTES / 4)
#define ROLL_HT_BYTES          960
#define ROLL_WD_PIXELS         960
#define ROLL_HT_PIXELS         960
#define ROLL_HT_RESOLUTION     600
#define ROLL_VT_RESOLUTION     600
#define ROLL_BIT_PERPIXEL      8


#define PLAIN_WD_BYTES         1600
#define PLAIN_WD_WORDS         (PLAIN_WD_BYTES / 4)
#define PLAIN_HT_BYTES         976
#define PLAIN_WD_PIXELS        1600
#define PLAIN_HT_PIXELS        976
#define PLAIN_HT_RESOLUTION    500
#define PLAIN_VT_RESOLUTION    500
#define PLAIN_BIT_PERPIXEL     8


#ifdef NEW_PALM
#define PALM_WD_PIXELS         2400              /* pixels per row */
#define PALM_HT_PIXELS         2040              /* rows */
#define PALM_HT_RESOLUTION     508
#define PALM_VT_RESOLUTION     508
#define PALM_BIT_PERPIXEL      8
#define HEEL_WD_PIXELS         2400              /* pixels per row */
#define HEEL_HT_PIXELS         1040              /* rows */
#define HEEL_HT_RESOLUTION     PALM_HT_RESOLUTION
#define HEEL_VT_RESOLUTION     PALM_VT_RESOLUTION
#define HEEL_BIT_PERPIXEL      8
#else
#define PALM_WD_BYTES          2748
#define PALM_WD_WORDS          (PALM_WD_BYTES / 4)
#define PALM_HT_BYTES          2748
#define PALM_WD_PIXELS         PALM_WD_BYTES
#define PALM_HT_PIXELS         PALM_HT_BYTES
#define PALM_HT_RESOLUTION     500
#define PALM_VT_RESOLUTION     500
#define PALM_BIT_PERPIXEL      8
#endif


/*
** ROLL: must be 0
```

- 67 -

```
#endif   /* DSP_MAP */

#endif   /* MEM_ALLC_H */
```

```
** GSP_COFF_PROGRAM will be stored after all tables, then followed by
** FINAL_IMAGE_ADDR.
*/
#define DSPGSP_CMD_BUF          0xC0002000       /* 0x1000 */


/*
** The order of the PC_PARAMETERS (ULONG):
** for Finger scanner
**      1. flag of the ROLL  equalization file exists (1=exists)
**      2. flag of the PLAIN equalization file exists (1=exists)
**      3. flag of the parameter file exists (1=exists)
**      4.... the numbers of the values in the para.dat file if exists
**
** for Palm scanner
**      1. flag of the PALM equalization file (palm.equ) exists (1=exists)
**      2. flag of the PALM dewarping file    (palm.wrp) exists (1=exists)
*/
#define PC_PARAMETERS           0xC0004000       /* 0x40   */


/*
** The general return results of GSP command processing, defined in the
** scancmd.h, will be stored after all image parameters in PC_PARAMETERS.
** If total numbers of the image parameters exceed to 0x40 in the future,
** this address needs to be changed.
*/
#define COMMAND_RETURN_RESULTS  0xC0004600       /* reserved 10W spaces */


/*
** below for finger scanner
*/
#define STRAIGHT_THRU_LUTTBL    0xC0004800       /* 0x4000   256*256/4 */
#define MINFUNC_LUTTBL          0xC0084800       /* 0x4000   256*256/4 */
#define FNGR_TAGGING_LUTTBL     0xC0104800       /* 0x4000   256*256/4 */
#define ROLL_EQU_REFTBL         0xC0184800       /* 0x38400  960*960/4 */
#define PLAIN_EQU_REFTBL        0xC088C800       /* 0x5F500 1600*976/4 */


/*
** CDRAM address in fngrgspe.cmd & fngrgspj.cmd need to be matched with
** FNGR_GSP_COFF_PROGRAM. If any one been changed, change the other, too
*/
#define FNGR_GSP_COFF_PROGRAM   0xC1476800       /* 0x7000 */
#define FNGR_FINAL_IMAGE_ADDR   0xC6000000


/*
** below for PALM scanner
*/
/*
** CDRAM address in palmgspe.cmd & palmgspj.cmd need to be matched with
** PALM_GSP_COFF_PROGRAM. If any one been changed, change the other, too
*/
#define PALM_EQU_REFTBL         0xC0004800       /* 2400/4 */
#define PALM_GSP_COFF_PROGRAM   0xC000C800       /* 0x7000  */


#ifndef NEW_PALM
#define PALM_FINAL_IMAGE_ADDR   0xC00EC800        /* 2748*2748/4=0x1CCE84 */
#else
#define PALM_DEWARP_TBL         0xC00EC800        /* 0x300 (3072/4) */
#define HEEL_FINAL_IMAGE_ADDR   0xC00F2800        /* 2400*1040/4=0x98580 */
#define PALM_FINAL_IMAGE_ADDR   0xC13FD800        /* 2400*2040/4=0x12ad60 */
#endif
```

```
**       1. flag of the ROLL  equalization file exists (1=exists)
**       2. flag of the PLAIN equalization file exists (1=exists)
**       3. flag of the parameter file exists (1=exists)
**       4.... the numbers of the values in the para.dat file if exists
**
** for Palm scanner
**       1. flag of the PALM equalization file (palm.equ) exists (1=exists)
**       2. flag of the PALM dewarping file   (palm.wrp) exists (1=exists)
*/
#define  PC_PARAMETERS          0xC00200      /* 0x40   */


/*
** The results of the platen check for the finger scanner, defined in the
** structure of platen_check_result_t in scancmd.h, will be stored after
** all image parameters in PC_PARAMETERS.
** If total numbers of the image parameters exceed to 0x40 in the future,
** this address needs to be changed.
*/
#define  COMMAND_RETURN_RESULTS  0xC00230      /* reserved 10W spaces */


/*
** below for Finger scanner
*/
#define  STRAIGHT_THRU_LUTTBL    0xC00240      /* 0x4000   256*256/4 */
#define  MINFUNC_LUTTBL          0xC04240      /* 0x4000   256*256/4 */
#define  FNGR_TAGGING_LUTTBL     0xC08240      /* 0x4000   256*256/4 */
#define  ROLL_EQU_REFTBL         0xC0C240      /* 0x38400  960*960/4 */
#define  PLAIN_EQU_REFTBL        0xC44640      /* 0x5F500 1600*976/4 */


/*
** GDRAM address in fngrgspe.cmd & fngrgspj.cmd need to be matched with
** FNGR_GSP_COFF_PROGRAM. If any one been changed, change the other, too
*/
#define  FNGR_GSP_COFF_PROGRAM   0xCA3840      /* 0x7000 */
#define  FNGR_FINAL_IMAGE_ADDR   0xF00000


/*
** below for PALM scanner
*/
/*
** GDRAM address in palmgspe.cmd & palmgspj.cmd need to be matched with
** PALM_GSP_COFF_PROGRAM. If any one been changed, change the other, too
*/
#define  PALM_EQU_REFTBL         0xC00240      /* 2400/4 */
#define  PALM_GSP_COFF_PROGRAM   0xC00640      /* 0x7000 */


#ifndef NEW_PALM
#define  PALM_FINAL_IMAGE_ADDR   0xC07640      /* 2748*2748/4=0x1CCE84 */
#else
#define  PALM_DEWARP_TBL         0xC07640      /* 0x300 (3072/4) */
#define  HEEL_FINAL_IMAGE_ADDR   0xC07940      /* 2400*1040/4=0x98580 */
#define  PALM_FINAL_IMAGE_ADDR   0xC9FEC0      /* 2400*2040/4=0x12ad40 */
#endif


#else   /* GSP_MAP */


/*
** Dram area to store the image process by WORD length:
** The first 2 items, DSPGSP_CMD_BUF, PC_PARAMETERS will be the same area
** to the finger scanner and the palm scanner.
```

- 70 -

```
/*
** Filename: mem_allc.h
** Purpose: Definition of memory allocation of software.
**          If the program wants to run GSP/RAMDAC/8051 functions under
**          the DSP map, add the compiler option -dDSP_MAP; otherwise,
**          those functions will run under GSP map.
** Date:    05/27/94
** Author:  Joyce Young
** History:
**   02/04/94 Joyce Young, Add the definitions of the GSP DRAM BANKs
**   03/15/94 Joyce Young, Rearrange the DramBanks
**   04/07/94 Joyce Young, Rearrange the DramBanks to support Kanji
**   05/20/94 Ellen Yu,    Removed unnecessary define's. Changed the
**                         FNGR_FINAL_IMAGE_ADDR from 0xE00000 to 0xF00000
**   06/07/94 Joyce Young, add COMMAND_RETURN_RESULTS to support platen check.
**   06/13/94 Joyce Young, modify the comments
**   11/22/94 JT, add #ifndef NEW_PALM to support new PALM
**   01/18/95 JT, modify for palm tables
*/


#ifndef MEM_ALLC_H
#define MEM_ALLC_H


/*
** Below memory maps are different from DSP and GSP.
** We use the same name on the both GSP and DSP codes, but
** compile with -dDSP_MAP option if the program is emulated on DSP or
** compile without -dDSP_MAP option if the program is emulated on GSP.
*/

#ifdef DSP_MAP


/*
** NOTE: finger scanner will use Bank0 & Bank3,
**       palm scanner will use all banks.
**       If any address is changed in mem_addr.h, make sure the address here
**       will be matched.
**
**                        +------------------+
** DramBank0:   0xC00100  | 1M words valid   |
**                        +------------------+
** DramBank1:   0xD00000  | 1M words valid   |
**                        +------------------+
** DramBank2:   0xE00000  | 1M words valid   |
**                        +------------------+
** DramBank3:   0xF00000  | 1M words valid   |
**                        +------------------+
*/


/*
** Dram area to store the image process by WORD length:
** The first 2 items, DSPGSP_CMD_BUF, PC_PARAMETERS will be the same area
** to the finger scanner and the palm scanner.
** GSP_COFF_PROGRAM will be stored after all tables, then followed by
** FINAL_IMAGE_ADDR.
*/
#define  DSPGSP_CMD_BUF          0xC00100        /* 0x100W */


/*
** The order of the PC_PARAMETERS (ULONG):
** for finger scanner
```

- 71 -

```
/*
** Filename: imgglobs.h
** Purpose:  Definition & variables to handle the images display or capture
**           on DSP/GEP.
** Date:     10/11/93
** Author:   Joyce Young
** Revised:
**   11/02/93 - Ellen Yu
**   06/07/94 - Ellen Yu, Support platen check function, add dirty_threshold
**              and damage_threshold to the image_object_t structure.
**   11/18/94 - JY, delete CAPT_ABORT in capture_t;
**              expand length of ic_name in image_object_t
**   12/21/94 - JY, add CAPT_NULL & CAPT_SAVE_PALM in capture_t
*/


#include "stdtypes.h"

#ifndef IMGGLOBS_H
#define IMGGLOBS_H

typedef enum capture_t
{
    CAPT_NO_RESPONSE,
    CAPT_SAVEING,       /* ROLL or PLAIN for finger, PALM & HEEL for palm */
    CAPT_MISSFNG,       /* missing finger/hand */
    CAPT_WRONG_TYPE,    /* imgage type and finger mismatched */
    CAPT_NULL,          /* 4 reserved for ADMIT_ABORT */
    CAPT_SAVE_PALM      /* save PALM only; only used by palm scanner */
) capture_t;


/*
** if MAX_NAME_LEN in scancmnd.h expands, ic_name needs to expand too.
*/
typedef struct image_object_t
{
    ULONG ic_cmd;           /* ROLL, PLAIN */
    ULONG ic_size;              /* sizeof (image_object_t)/sizeof (ULONG) */
    ULONG ic_scantype;      /* STYPE_... */
    ULONG ic_imgtype;       /* ROLL, PLAIN or PALM */
    ULONG ic_set;           /* OR of shifted image IDs */
    ULONG ic_addr;          /* the address of image */
    ULONG ic_name[11];/* convict name (4 chars/ULONG) */
    ULONG ic_reserve1;
         ULONG ic_dirty_threshold;
         ULONG ic_damage_threshold;
    ULONG ic_reserve2;
    ULONG ic_flag;          /* the flag indicate all words written */
) image_object_t;


#endif   /* IMGGLOBS_H */
```

```
/*
** Filename: gsp_func.h
** Purpose:  function prototypes
** Date:     04-19-95
** Author:   Ellen Yu
** History:
*/

/* .......................... grabber.c ..........................*/
VOID init_gsp_grabber (VOID);
VOID turn_grab_off (image_type_t camera);
VOID turn_grab_on (image_type_t camera, int frame_num);
VOID turn_grab_on_normal (image_type_t camera, int frame_num);
VOID wait_for_grab_VBLANK_on_off (VOID);
```

```
/* $Header:   G:/t1000/gsp/mimio.h_v   1.0   08 Apr 1995 11:06:02   TOM  $ */
/*
 * $Log:   G:/t1000/gsp/mimio.h_v  $
 *
 *    Rev 1.0   08 Apr 1995 11:06:02   TOM
 * Initial revision.
 *
 *    Rev 1.2   26 Oct 1992 17:44:50   TOM
 * added mim_fill declarations
 *
 *    Rev 1.1   16 Oct 1992 01:52:10   TOM
 *
 *    Rev 1.0   16 Sep 1992 18:41:04   TOM
 * Initial revision.
 */
/*********************************************************
file: mimio.h

definitions for memory image segments
only 8-bit pixels are allowed

*/

typedef struct mimage {
    unsigned char *a;       /* address of first element of first row */
    int    i;               /* increment in bytes  from row to row  */
    int    w;               /* image width */
    int    h;               /* image height */
) MIM;

void mim_new(MIM *d,unsigned char *a,int i,int w,int h );
void mim_subset( MIM *s,MIM *d,int x,int y,int w,int h );
void mim_move( MIM *s,MIM *d );
void mim_move_c( MIM *s MIM *d,int c);
void mim_op_move( MIM *s,MIM *d );
unsigned char *mim_adr( MIM *s,int x,int y );
void mim_size( MIM *s, int *w, int *h );
int mim_inc( MIM *s);
void mim_fill( MIM *d, int v );
void mim_fill_c( MIM *d, int v, int c );
```

```
/*
** Filename: gsp_imgs.h
** Purpose: Definition & variables to handle the images display or capture.
** Date:    18/01/94
** Author:  Ellen Yu
** History:
** 02/20/94 Joyce Young, modify the MINORAM offset
** 06/03/94 Joyce Young, adjust the offset of ROLL/PLAIN active image in
**                MINORAM for new board
** 06/09/94 Ellen Yu, add #define PROTOTYPE difference the hardware
**                prototype board and other new revisions.
*/


#ifndef GSP_IMGS_H
#define GSP_IMGS_H


#define MINORAM_WD_BYTES      2048
#define MINORAM_WD_WORDS      (MINORAM_WD_BYTES / 4)
#define MINORAM_HT_BYTES      1024
#define MINORAM_PITCH_BYTES   2048
#define MINORAM_PITCH_BITS    (MINORAM_PITCH_BYTES * 8)    /* 0x4000 */
#ifndef PROTOTYPE
#define MINORAM_ROLL_ROW_B_OFFSET   38
#define MINORAM_ROLL_COL_W_OFFSET   24
#define MINORAM_ROLL_COL_B_OFFSET   (MINORAM_ROLL_COL_W_OFFSET * 4)
#define MINORAM_PLAIN_ROW_B_OFFSET  31
#define MINORAM_PLAIN_COL_W_OFFSET  38
#define MINORAM_PLAIN_COL_B_OFFSET  (MINORAM_PLAIN_COL_W_OFFSET * 4)
#else
#define MINORAM_ROW_B_OFFSET 32
#define MINORAM_COL_W_OFFSET 24
#define MINORAM_COL_B_OFFSET (MINORAM_COL_W_OFFSET * 4)
#define MINORAM_PLAIN_COL_W_OFFSET 13
#define MINORAM_PLAIN_COL_B_OFFSET (MINORAM_PLAIN_COL_W_OFFSET * 4)
#endif


#define GRBVRAM_WD_BYTES      1024
#define GRBVRAM_HT_BYTES      1024
#define GRBVRAM_PITCH_BYTES   1024
#define GRBVRAM_PITCH_BITS    (GRBVRAM_PITCH_BYTES * 8)    /* 0x2000 */


#define OPTVRAM_WD_BYTES      1024
#define OPTVRAM_HT_BYTES      1024
#define OPTVRAM_PITCH_BYTES   1024
#define OPTVRAM_PITCH_BITS    (1024 * 8)    /* 0x2000 */


#define ROLL_PITCH_BYTES      1024
#define ROLL_PITCH_BITS       (ROLL_PITCH_BYTES * 8)
#define ROLL_GRBVRAM_PITCH    (-(GRBVRAM_PITCH_BITS * 4))
#define ROLL_OPTVRAM_PITCH    (OPTVRAM_PITCH_BITS * 3)
#define ROLL_DYDX_VAL         (((((IMG_ROLL_H - 12) / 3) << 16) | (IMG_ROLL_W - 16))


#define PLAIN_PITCH_BYTES     2048
#define PLAIN_PITCH_BITS      (PLAIN_PITCH_BYTES * 8)
#define PLAIN_GRBVRAM_PITCH   (-(GRBVRAM_PITCH_BITS * 2))
#define PLAIN_OPTVRAM_PITCH   (OPTVRAM_PITCH_BITS)
#define PLAIN_DYDX_VAL        (((IMG_PLAIN_H - 12) << 16) | (IMG_PLAIN_W - 12))


#endif    /* GSP_IMGS_H */
```

```
    ERASE,
    DRAW
) TEXT_OPER_T;

typedef enum
(
    SCAN_KEY,
    SAVE_KEY
) KEY_TYPE;

typedef enum
(
    RET_ERR_ADDR,
    READ_FOREVER,
    WRITE_FOREVER
) MEM_TEST_RESULT_TYPE;

typedef enum
(
    UNPRESSED_BUTTON,
    PRESSED_BUTTON
) BUTTON_OPER_TYPE;


/*
** Attributes for text displayed on screen
*/
typedef struct
(
    short    font;        /* font index of the talon_font table */
    short    fcolor;      /* foreground color */
    short    bcolor;      /* background color */
    short    xposn;       /* x position */
    short    yposn;       /* y position */
    short    align;       /* Alignment (relative to baseline or topleft) */
    short    font_h;      /* height of font */
    short    text_w;      /* width of text string */
    char     *strptr;     /* text string ptr */
) SCRTEXT_T;

typedef struct
(
    image_type_t  camera;
    HAND_TYPE     hand;
    scan_type_t   fng_num;
    ULONG         store_addr;
    UBYTE         *src_addr;
    UBYTE         *dat_addr;
) IMAGE_ATTR_T;


#endif   /* GSP_DEFS_H */
```

```
typedef enum
(
    PREV_ROLL,
    PREV_PLAIN,
    PREV_MISSING,
    PREV_YES,
    PREV_NO,
    PREV_MISFNG,
    CAPT_REPEAT,
    CAPT_NEXT,
    CAPT_FINGER,
    PLACE_START,
    PLACE_ABORT,
    ROLL_CONTINUE,
    ROLL_YES
) KEYACT_TYPE;

typedef enum
(
        RIGHT,
        LEFT
) HAND_TYPE;

typedef enum
        (
        ROLL_RIGHT_THUMB,               /* 0 */
        ROLL_RIGHT_INDEX,               /* 1 */
        ROLL_RIGHT_MIDDLE,              /* 2 */
        ROLL_RIGHT_RING,                /* 3 */
        ROLL_RIGHT_LITTLE,              /* 4 */
        ROLL_LEFT_THUMB,                /* 5 */
        ROLL_LEFT_INDEX,                /* 6 */
        ROLL_LEFT_MIDDLE,               /* 7 */
        ROLL_LEFT_RING,                 /* 8 */
        ROLL_LEFT_LITTLE,               /* 9 */
        PLAIN_TWO_THUMBS,               /* 10 (unsupported) */
        PLAIN_RIGHT4,                   /* 11 */
        PLAIN_LEFT4,                    /* 12 */
        RIGHT_PALM,                     /* 13 */
        LEFT_PALM,                      /* 14 */
        PLAIN_RIGHT_THUMB,              /* 15 */
        PLAIN_LEFT_THUMB                /* 16 */
) FINGER_TYPE;

/*
** user-selected type of the key pressed or command for image caputre
*/
typedef enum keytype_t
(
    TYPE_NO_RESPONSE,
    TYPE_SCAN_KEY,
    RELEASE_SCAN_KEY,
    TYPE_SAVE_KEY,
    RELEASE_SAVE_KEY,
    TYPE_ABORT
) keytype_t;

typedef enum
(
```

```
/*
** Filename: gsp_defs.h
** Purpose:  #define in GSP application.
** Date:     10-05-93
** Author:   Ellen Yu
** History:
** 01-31-94 Joyce Young, Changed the src_addr & dst_addr in IMAGE_ATTR_T as
**                       pointer
** 02-25-94 Joyce Young, Added the fng_num in IMAGE_ATTR_T
** 04-01-94 Joyce Young, support KANJI in MODE_TYPE
** 12-19-94 Ellen Yu, remove some unused define
*/

#ifndef GSP_DEFS_H
#define GSP_DEFS_H

#include "img_defs.h"
#include "scancmd.h"

#define NULL    0

#define FALSE   0
#define TRUE    1

/*
**
*/
#define X1E     0x0002
#define X1E     0x0200
#define INTIN   0x0008
#define INTOUT  0x0080

#define BIT0    0x01
#define BIT1    0x02
#define BIT2    0x04
#define BIT3    0x08

/*
**
*/
typedef enum
{
    MODE_ROLL,
    MODE_PLAIN,
    MODE_PREV,
    MODE_CENTR,
    MODE_RARROW,
    MODE_LARROW,
    MODE_PLACEHOLD,
    MODE_PLACEHOLD1,
    MODE_ROLLHOLD,
    MODE_ROLLHOLD1,
    MODE_LIFTHAND
} MODE_TYPE;

typedef enum
{
    FINGER_LEGEND,
    MODE_LEGEND
} WGS_LEGEND_TYPE;
```

```
/*
** wait for Grab is enabled (BIT0 becomes 0)
*/
while ( ((*(UBYTE *)GRABBER_CTRL0_BASE) & 1) )
    ;

for ( ii = 0; ii < frame_num; ii++ )
    {
    while ( !((*(UBYTE *)GRABBER_CTRL0_BASE) & 2) )
        ;

    while ( ((*(UBYTE *)GRABBER_CTRL0_BASE) & 2) )
        ;
    }
#endif
} /* turn_grab_on_normal */


/* ...............................................................*/
/*
** wait for the VBLANK start on and off
*/
VOID wait_for_grab_VBLANK_on_off (VOID)
{
#ifndef DISABLE_GRAB
    while ( !((*(UBYTE *)GRABBER_CTRL0_BASE) & 2) )
        ;

    while ( ((*(UBYTE *)GRABBER_CTRL0_BASE) & 2) )
        ;
#endif
} /* wait_for_grab_VBLANK_on_off */


/* end of grabber.c */
```

- 79 -

The present invention has been described in terms
of preferred embodiments. The invention, however, is not
limited to the embodiments depicted and described.
Rather, the scope of the invention is defined by the
5 appended claims.

-80-

CLAIMS;

1   1.   A method of reducing smear in a rolled

2   fingerprint image represented by a rolled image array,

3   comprising the steps of:

4   sequentially generating frames of an optical

5   image signal which includes data values characteristic of

6   light intensities of corresponding locations of an

7   optical image, wherein the optical image includes a

8   fingerprint image of a finger rolling on a surface;

9   determining, for each frame of the optical

10   image signal, a freeze column representing a line

11   positioned between leading and trailing edges of the

12   fingerprint image and oriented transverse to a direction

13   of roll of the rolling finger;

14   sequentially updating an interim array that

15   is an accumulation of the frames of the optical image

16   signal and characteristic of an interim image of a rolled

17   fingerprint, a current update of the interim array being

18   formed by reducing pixel values of the interim array by a

19   portion of the difference between corresponding data

20   values from a current frame of the optical image signal

21   and the pixel values of the interim array if the

22   corresponding data values of the current frame of the

23   optical image signal are less than the pixel values of

24   the interim array; and

25   generating the rolled image array by

26   transferring portions of the interim array to the rolled

27   image array, wherein the transferred portion of the

28   current update of the interim array extends forward in

29   the direction of finger roll from the freeze column

30   determined from a preceding frame of the optical image

31   signal that preceded the current frame of the optical

32   image signal.

1   2.   The method of claim 1, wherein the

2   transferred portion of the current update of the interim

- 81 -

3   array includes data characteristic of a portion of the
4   interim image up to approximately the leading edge of the
5   rolled fingerprint of the interim image.

1   3.   The method of claim 1, wherein the
2   transferred portion of the current update of the interim
3   array extends up to approximately the freeze column
4   determined from the current frame of the optical image
5   signal.

1   4.   The method of claim 1, wherein the freeze
2   line represented by the freeze column determined from
3   each frame of the optical image signal is positioned at
4   least approximately half a distance in the direction of
5   roll between the leading and trailing edges of the
6   fingerprint image.

1   5.   The method of claim 1, wherein the freeze
2   line represented by the freeze column determined from
3   each frame of the optical image signal is positioned more
4   than half a distance in the direction of roll between the
5   leading and trailing edges of the fingerprint image.

1   6.   The method of claim 1, wherein a first
2   transferred portion of the interim array extends rearward
3   in the direction of finger roll from approximately the
4   freeze column determined from a first frame of the
5   optical image signals.

1   7.   The method of claim 6, wherein the first
2   transferred portion of the interim array is
3   characteristic of a portion of the interim image forward
4   in the direction of roll from about the trailing edge of
5   the first interim image.

- 82 -

1   8.   The method of claim 1, further comprising
2   initializing pixels in the interim array with maximum
3   values.

1   9.   The method of claim 1, further comprising
2   initializing the interim array with data values of a
3   frame of the optical image signal.

1   10.   The method of claim 1, further comprising
2   saving the rolled image array after transferring the
3   portion of a final interim array to the rolled image
4   array.

1   11.   The method of claim 1, further comprising
2   displaying a rolled fingerprint image represented by the
3   rolled image array on a display device as it is
4   generated.

1   12.   The method of claim 11, further comprising
2   decimating the transferred portion such that the rolled
3   image array has fewer pixels than the interim array.

1   13.   The method of claim 1, wherein sequentially
2   updating the interim array includes updating the interim
3   array in real time as frames of the optical image signal
4   are generated, and wherein generating the rolled image
5   array includes transferring a portion of the interim
6   array to the rolled image array in real time as the
7   interim array is updated.

1   14.   The method of claim 1, wherein the
2   transferred portions of the interim array are adjacent
3   and non-overlapping.

- 83 -

1   15. A method of generating a rolled fingerprint

2   image array characteristic of a rolled fingerprint image,

3   comprising the steps of:

4   generating a series of frames of an optical

5   image signal characteristic of an optical image of a

6   finger rolling on a surface at sequential times, wherein

7   the frames include data, the value of each datum being

8   characteristic of a light intensity of a corresponding

9   location of the optical image of the rolling finger;

10   determining a freeze column from each frame,

11   wherein each freeze column is representative of a

12   position between leading and trailing edges of the

13   corresponding optical image of the rolling finger;

14   sequentially updating an interim array in an

15   image memory with the frames as they are generated, the

16   interim array being characteristic of an interim image of

17   a rolled fingerprint that has a leading edge and a

18   trailing edge, including first updating the interim array

19   by transferring a first one of the optical image signals

20   to the image memory, and then further updating the

21   interim array by reducing pixel values of the interim

22   array with a portion of the difference between

23   corresponding data values of a current frame and the

24   pixel values of the interim array if the corresponding

25   data values of the current frame indicate a darker image

26   than the pixel values of the interim array;

27   associating the freeze columns determined

28   from the frames with corresponding updates of the interim

29   array; and

30   ·   sequentially updating the rolled fingerprint

31   image array in an output memory with the updates of the

32   interim array by sequentially transferring a portion of

33   each update of the interim array to the output memory,

34   including transferring a portion of the first update of

35   the interim array that extends in a direction of finger

- 84 -

36 roll rearward from approximately the freeze column
37 associated with the first updated interim array, and then
38 transferring a portion of a subsequent update of the
39 interim array that extends forward in the direction of
40 finger roll from approximately the freeze column
41 associated with a preceding update of the interim array.

1   16. The method of claim 15, wherein the
2   transferred portion for a subsequent update of the
3   interim array extends forward only to the freeze column
4   of the subsequently updated interim array.

1   17. The method of claim 15, wherein the
2   transferred portions of sequential updates of the interim
3   array are adjacent and non-overlapping.

1   18. The method of claim 15, wherein the
2   transferred portions of sequential updates of the interim
3   array are adjacent and do not overlap rearward in the
4   direction of finger roll from the freeze column
5   determined from the preceding optical image signal.

1   19. A device for reducing smear in a rolled
2   fingerprint image represented by a rolled image array,
3   comprising:
4   an imaging system for sequentially generating
5   frames of a series of electronic signals characteristic
6   of an optical image that includes a fingerprint image of
7   a finger rolling on a surface;
8   means for sequentially generating frames of
9   an optical image signal in response to the electronic
10 signals, each optical image signal including data, the
11 value of each datum being characteristic of a light
12 intensity of a corresponding location of the optical
13 image;

- 85 -

14   an image capture system responsive to the
15 optical image signals for sequentially updating an
16 interim array characteristic of an interim image of a
17 rolled fingerprint that has a leading edge and a trailing
18 edge, a current update of the interim array being formed
19 from a preceding update of the interim array and a
20 current frame of the optical image signal by reducing
21 pixel values of the preceding update of the interim array
22 with a portion of the difference between the
23 corresponding data values of the current frame and the
24 pixel values of the preceding update of the interim array
25 if the corresponding data values of the current frame are
26 characteristic of darker images than the pixel values of
27 the preceding update of the interim array;
28   means for determining, for each frame of the
29 optical image signal, a freeze column representing a line
30 positioned between leading and trailing edges of the
31 fingerprint image and oriented transverse to a direction
32 of roll of the rolling finger; and
33   means for generating the rolled image array
34 by transferring a portion of the current interim array to
35 the rolled image array, wherein the transferred portion
36 of the current interim array extends forward in the
37 direction of finger roll from the freeze column
38 determined from a preceding frame of the optical image
39 signal that preceded the current frame of the optical
40 image signal.

1   20. The method of claim 1, wherein the current
2 update of the interim array F^n is formed according to the
3 relation:
4                  $F^n = F^{n-1} - K*(F^{n-1} - I^n)$,
5 where F^n is a pixel value of the current update of interim
6 array, F^{n-1} is the pixel value of interim array, I^n is the

- 86 -

7 corresponding data value of the optical image signal, and

8 K is a factor less than or equal to one.

1   21.   The method of claim 20, wherein K is in a

2 range of 0.25 to 0.5.

1   22.   The method of claim 20, wherein K is

2 approximately 0.33.
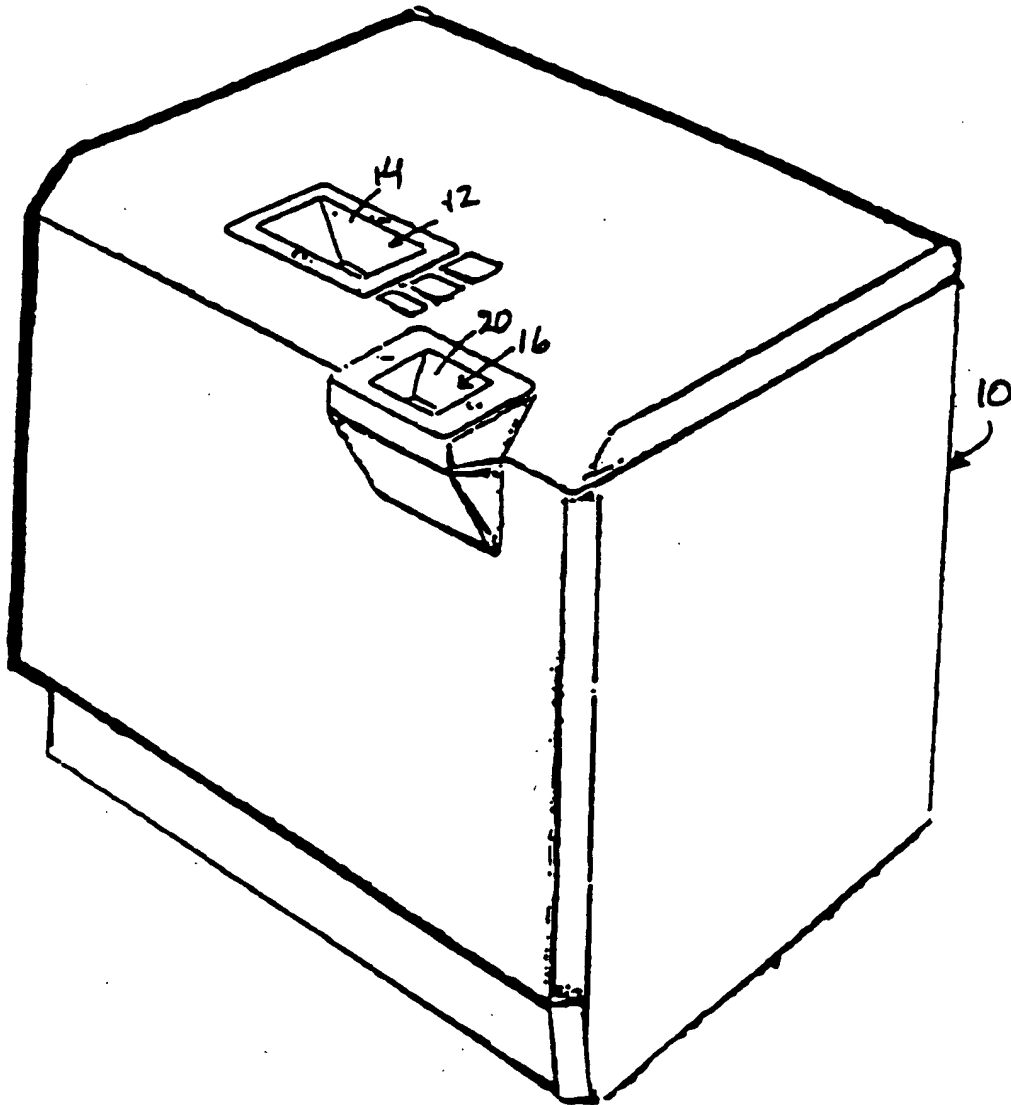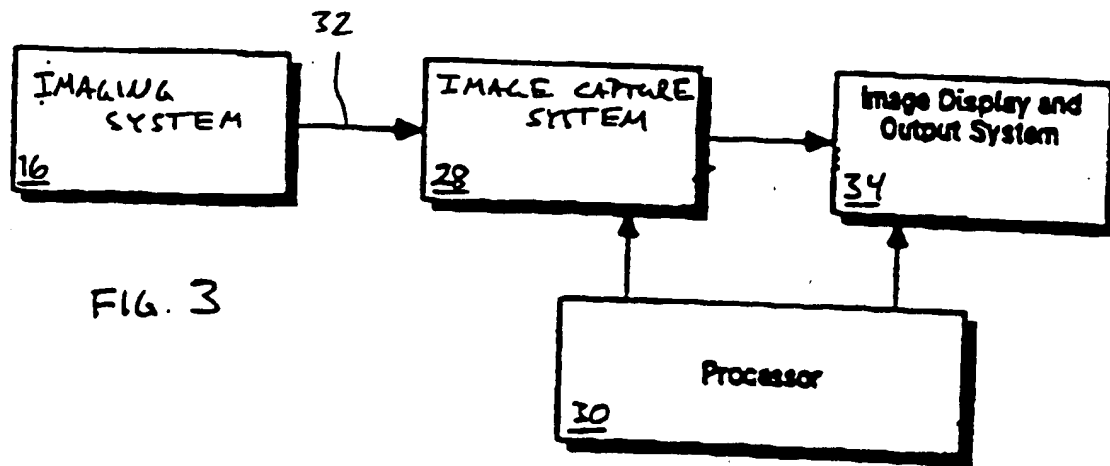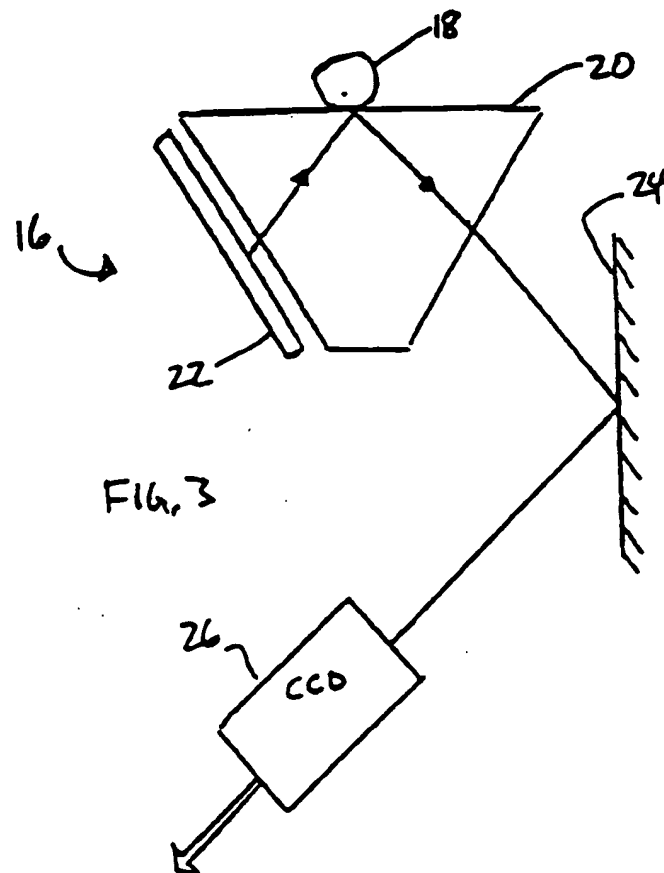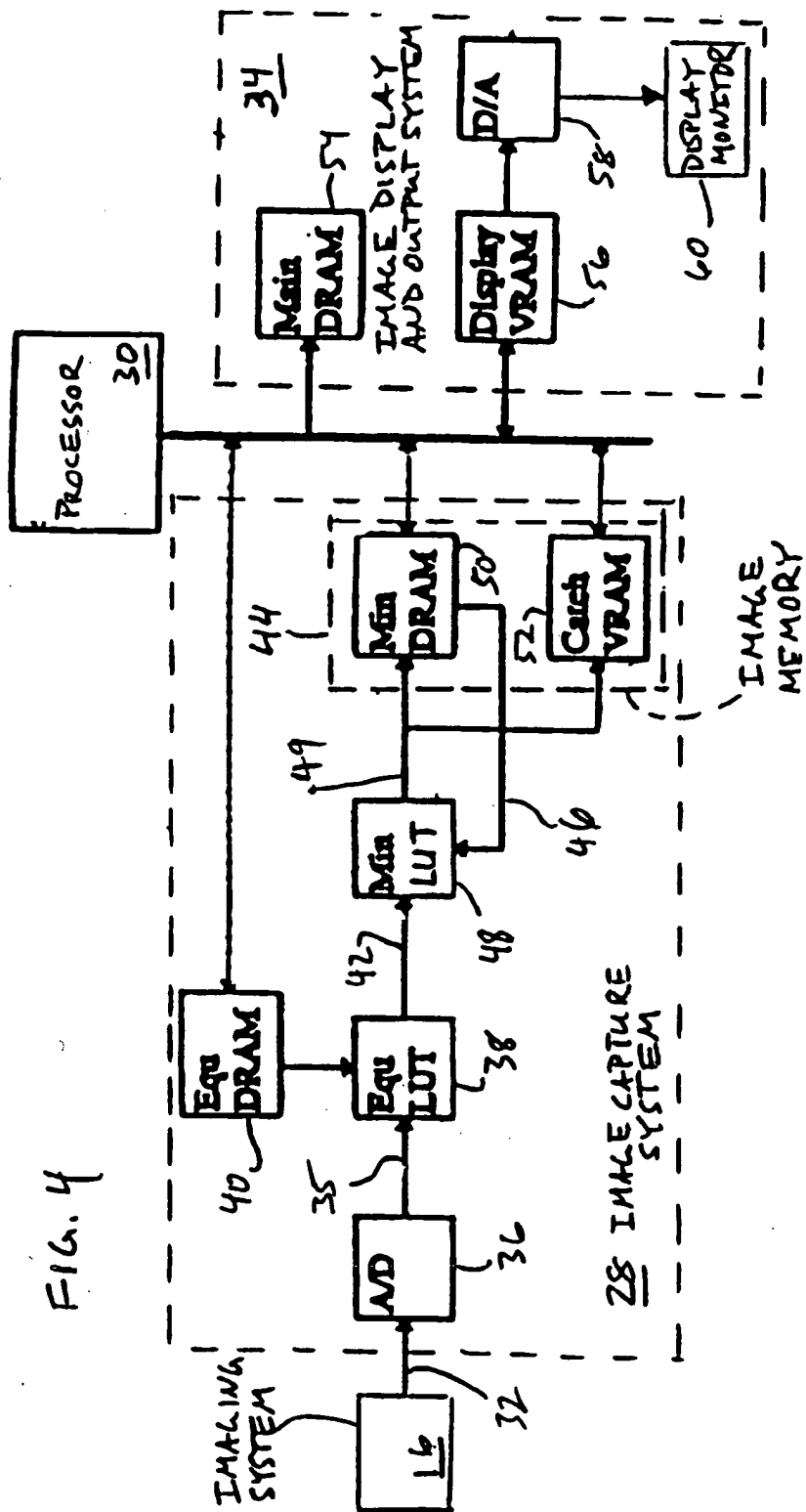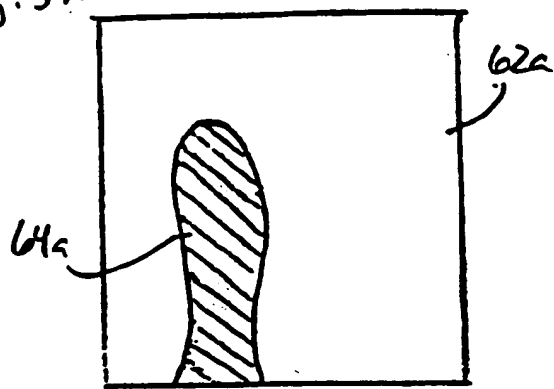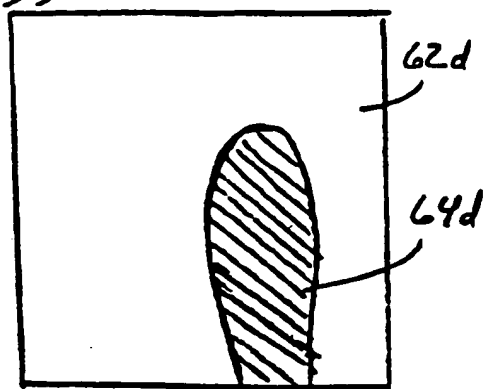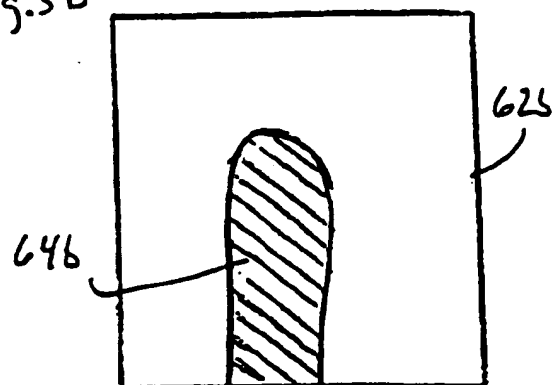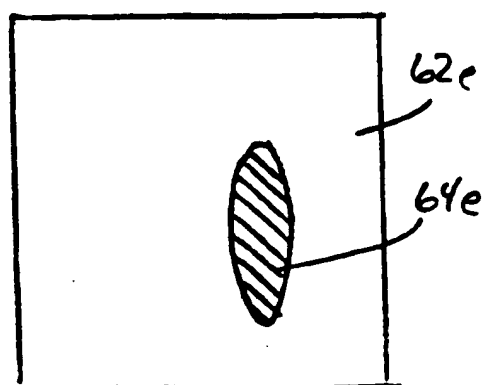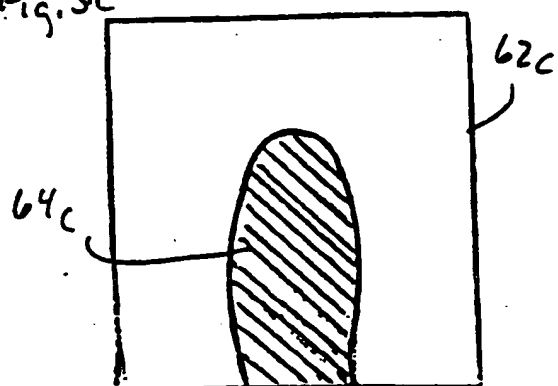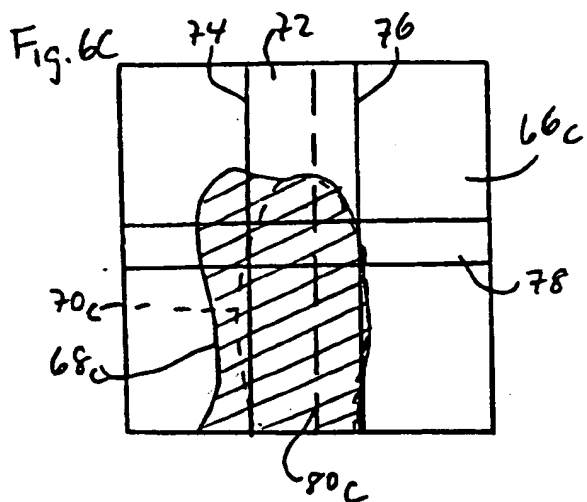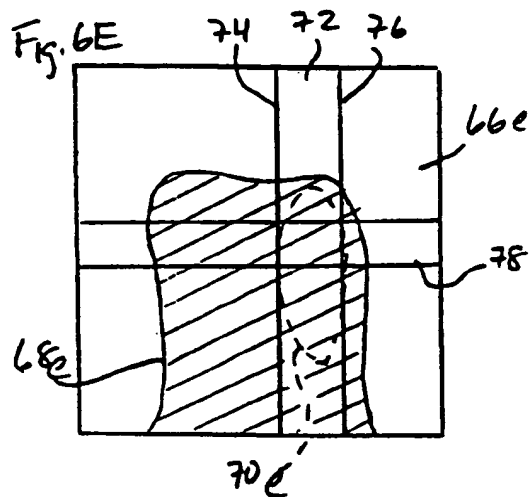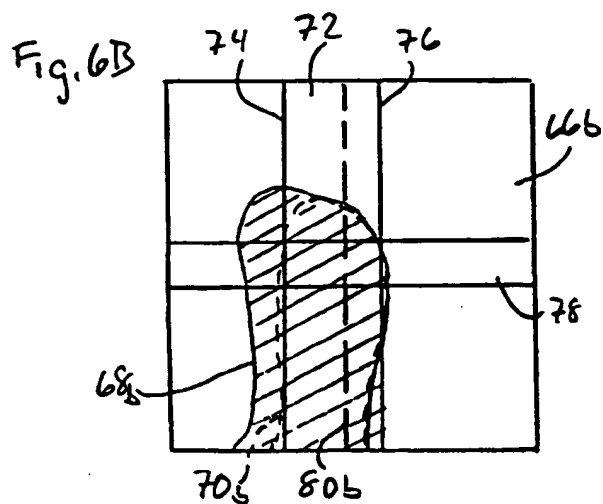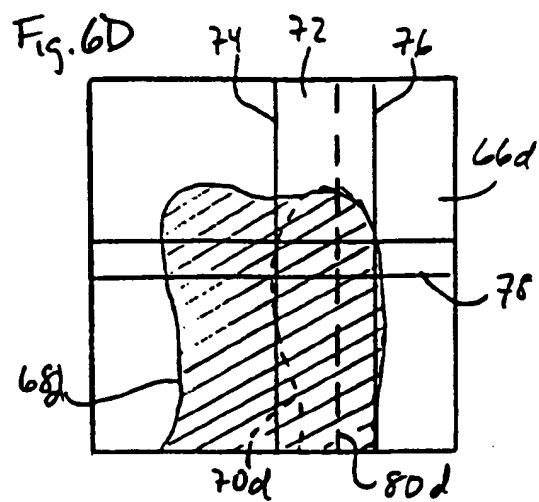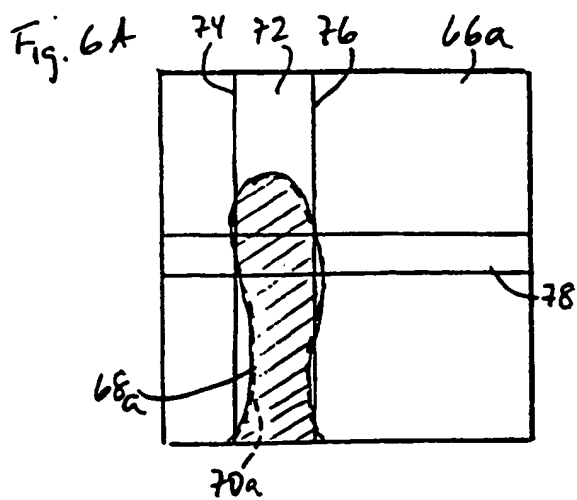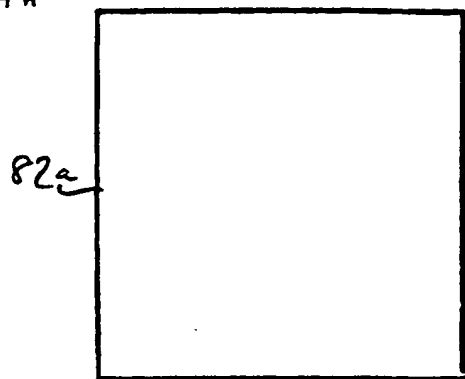
Fig. 1

IMAGING SYSTEM 16 → IMAGE CAPTURE SYSTEM 28 → Image Display and Output System 34

Processor 30

FIG. 3

16

18

20

22

24

26 CCD

FIG. 3

FIG. 2
PRIOR ART

FIG. 4

Fig. 5A

62a

64a

Fig. 5D

62d

64d

Fig. 5B

62b

64b

Fig. 5E

62e

64e

Fig. 5C

62c

64c

Fig. 6A

74 72 76   66a

68a

70a

78

Fig. 6B

74   72   76

66b

68b

70b   80b

78

Fig. 6C

74   72   76

66c

70c

68c

80c

78

Fig. 6D

74   72   76

66d

68d

70d   80d

78

Fig. 6E

74 72 76

66e

68e

70e

78

Fig. 7A

82a

Fig. 7D   80c   84d

82d

80d

Fig. 7B   80b

82b

84b

Fig. 7E   80d

82e   84e

Fig. 7C   80b   84c

82c

80c

Fig 8A

86a

Fig. 8d   80c

76

86d

88d

Fig. 8B   80b

86b

88b

Fig. 8e   80d

86e   88e

Fig. 8c   80b

76

86c

88c
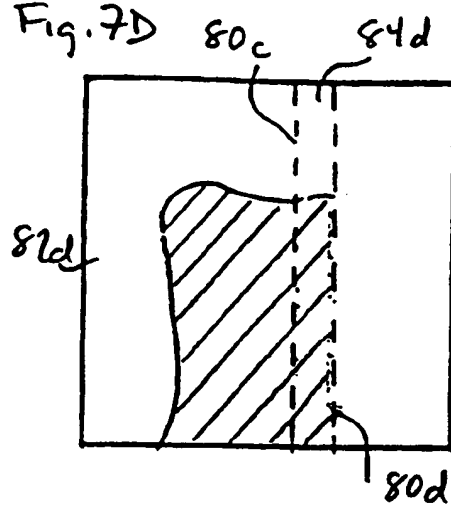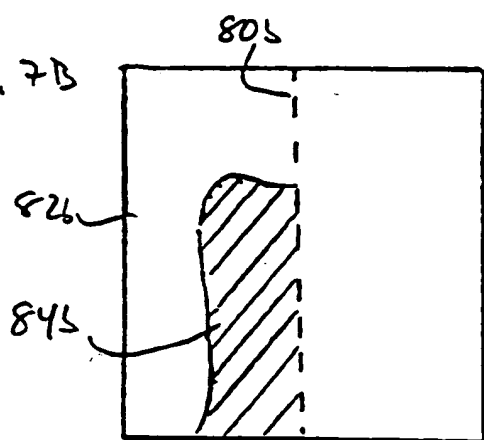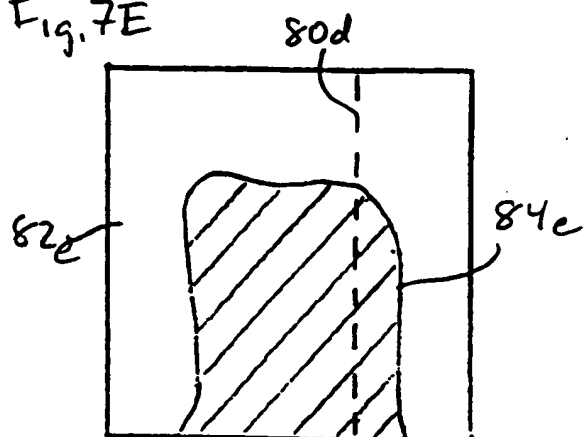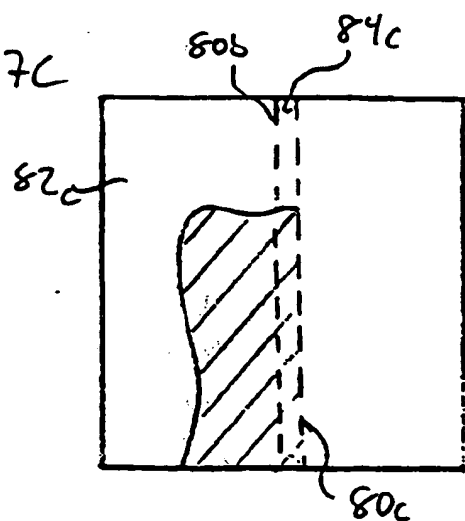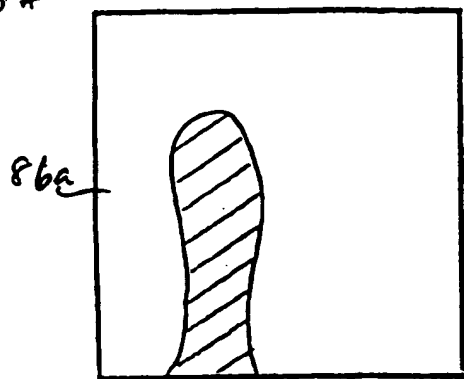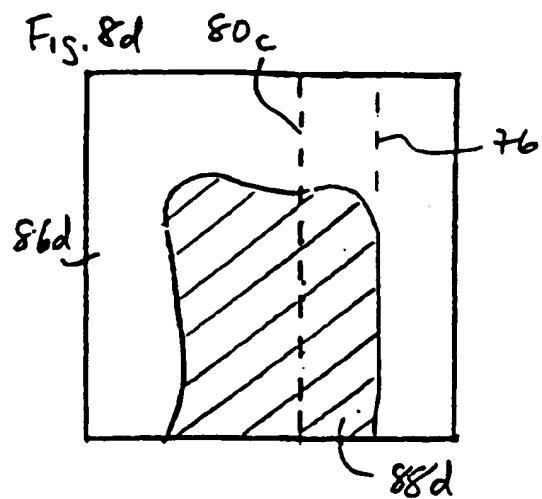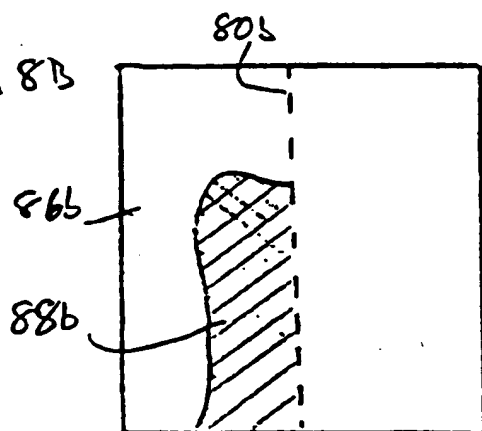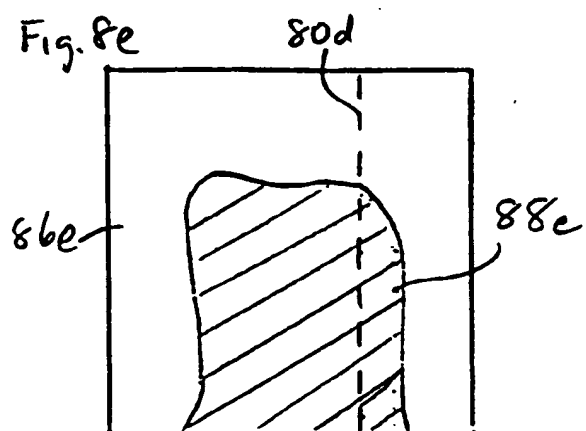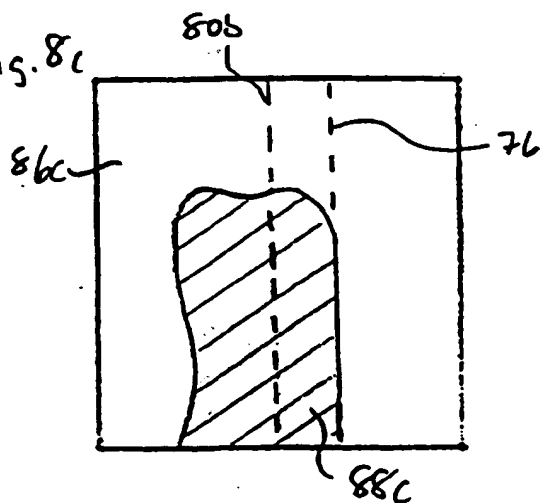
A. CLASSIFICATION OF SUBJECT MATTER

IPC(6) :G06K 9/00; G06K 9/74

US CL :382/124; 356/71

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

U.S. : 382/124, 125, 126; 356/71

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

APS, MAYA

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	US 5,230,025 A (FISHBINE ET AL) 20 July 1993, col. 7, lines 14-53	1-22
A	US 4,553,837 A (MARCUS) 19 November 1985.	1-22
A, P	US 5,548,394 A (GILES ET AL) 20 August 1996.	1-22
A	US 4,933,976 A (FISHBINE ET AL) 12 June 1990.	1-22

☐ Further documents are listed in the continuation of Box C.   ☐ See patent family annex.

• Special categories of cited documents:	"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"A" document defining the general state of the art which is not considered to be part of particular relevance	
"E" earlier document published on or after the international filing date	"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)	"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
"O" document referring to an oral disclosure, use, exhibition or other means	
"P" document published prior to the international filing date but later than the priority date claimed	"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
03 JULY 1997	2 0 AUG 1997

Name and mailing address of the ISA/US Commissioner of Patents and Trademarks Box PCT Washington, D.C. 20231	Authorized officer BHAVESH MEHTA
Facsimile No. (703) 305-3230	Telephone No. (703) 308-5246

Form PCT/ISA/210 (second sheet)(July 1992)*